IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 6.252

Volume 2, Issue 2, March 2022

Study On Go Programming Language

Shrunga G S[1], Sindhu R[2], Sneha U B[3], Soumya Ashok Malakannavar[4], Deepika Kamath[5]

Student, Department of Computer Science and Engineering[1,2,3,4]

Guide and Assistant Professor, Department of Computer Science and Engineering[5]

Alva's Institute of Engineering and Technology, Mangalore, India

Abstract: *When developing software today, we still use old tools and ideas. Maybe it is time to start from scratch and try tools and languages that are more in line with how we actually want to develop software. The Go Programming Language was created at Google by a rather famous trio: Rob Pike, Ken Thompson and Robert Griesemer. Before introducing Go, the company suffered from their development process not scaling well due to slow builds, uncontrolled dependencies, hard to read code, poor documentation and so on. Go is set out to provide a solution for these issues. The purpose of this master's thesis was to review the current state of the language. This is not only a study of the language itself but an investigation of the whole software development process using Go. The study was carried out from an embedded development perspective which includes an investigation of compilers and cross-compilation. We found that Go is exciting, fun to use and fulfills what is promised in many cases. However, we think the tools need some more time to mature.*

Keywords: Go, Golang, Language Review, Cross-Compilation, Developer Tools, Embedded

I. Introduction

In programming language discussions a quote from Lawrence Flon often shows up "There does not now, nor will there ever, exist a programming language in which it is least bit hard to write bad programs."

This is important to have in mind when trying a new programming language: we cannot expect it to be perfect because a bad programmer will always find a way of misusing the language structures. We should instead focus on how the language helps developers in using good coding practices. Go cannot let us do things that are not possible with other languages but the question is how the language lets us do it. For example thread synchronization can be achieved in Java but maybe it is easier to do in Go.

II. The Go Programming Language

Go has been described as "the C for the 21st century" and that it makes you feel like "being young again (but more productive!)". It is a compiled and statically typed language with C-like but simpler syntax and garbage collection. It was designed and developed to meet certain problems experienced by software engineers at Google.

This company typically used Java, C++ and Python for their large projects and some of their projects they claimed to be indeed very large which makes some problems with the development cycle more evident. Some of the major points were:

1. Build time not scaling well, which slow down the whole development cycle, partly caused by
2. Limited code and package understandability in the team augmented by
3. The usage of different language subsets and patterns among developers (e.g. in C++)leading to
4. Unnecessary safe guarding package imports that again augments confusion and slows down builds with languages like C where files typically has to be opened before reaching a header guard.

Go was designed specifically to address these points through:

1. Reduced build time with a model for dependency management.
2. Reduced bugs arising due to no pointer arithmetic2 and by using a run-time garbage collector.
3. No complicated type system.
4. A concise language specification that has few keywords and lacks complicated constructs.
5. Language has built-in and easy to use mechanisms for concurrency.

One important thing to note here, which is also stressed by the creators of Go, is that concurrency is not parallelism. Concurrency is about organizing the code such that different parts can run at the same time along with synchronization and communication between the parts. Parallelism is about actually running things at the same time. This means that concurrency enables parallelism.

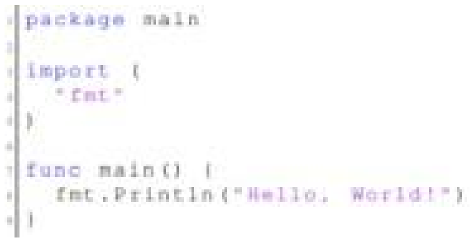
The built-in mechanism for concurrency is revolved around the so called goroutine. They function as lightweight threads that are handled by the Go runtime and it is cheap to start many of them (in scale of thousands). There is also a built-in type called channel that enables safe communication and synchronizations between goroutines. The most important thing to study is obviously the language syntax, features and standard library.

- Is the language itself easy to use and understand when having a background similar to ours i.e. being adept in languages like C, C++ or Java?
- Is it easier and less error prone to write correct concurrent software in Go compared to other mainstream languages?
- Are the standard libraries and other common libraries mature enough for usage in production software development?
- Before learning and using a language there should be some type of assurance of the future and maintenance of the language, libraries, compilers and tools. Go is developed by Google, but how is the language governed now (and in the future)?
- How much can other parties influence the development of Go and what would happen if Google loses interest; is there a committee and community that could takeover?
- Is a permissive license in use for the language specification and current implementation of tools and libraries?

III. Syntax and Types

The syntax will be familiar for programmers used to languages from the C-syntax family but it is clean and resembles how easy it is to write Python code. To start off, study the hello world program in listing

Hello World



```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, World!")
}
```

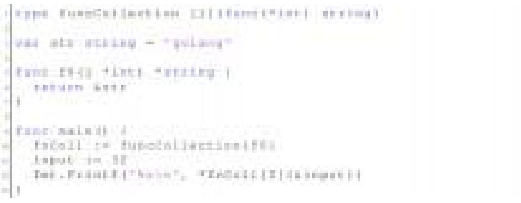
First the package is specified, in this case the main package since this package will contain a main function. After that we have a list of imported packages, in this case the standard string formatting package. The imported package can either refer to a local package on the system or can be a URL as described in section 2.3.5. Then we see a declaration of the main function comes, with no arguments and no return value. As can be seen, statements in Go are not semicolon terminated1. Also notice that type visibility is determined by the first letter of the name like the Println() function from the fmt package. Type names beginning with a capital letter are exported whereas lower cased names are not visible outside the defining scope.

3.1 Types in Go

1. var i int // All Go types have sensible zero values , 0 here .
2. j := 3.14 // Type is inferred , float64 on our machine .
3. str0 := " gopher "
4. var str1 * string = & str0 // str is a pointer to a string

Declaring types from left-to-right feels strange in the beginning for a C-programmer, but it does not take long before it becomes natural. To further illustrate how much of a difference this means for complex types, compare the two equivalent programs written in C and Go in listing 2.3 and 2.4. They declare (at line number 1) a new type that is an array of size one of functions that takes a pointer to integer and returns a string. It is worth to notice that in Go we do not take the address of a function as functions are "first-class" i.e. directly supported as a value.

Complicated type in Go



```
type funcCollection [][](func(*int) string)

var str string = "golang"

func f4() *int) *string {
  return &str
}

func main() {
  fnColl := funcCollection{f4}
  input := 5
  fmt.Printf("%s\n", *fnColl[0](&input))
}
```

3.2 Object-Orientation

Since Go is a modern language, we would expect the common Object- orientation2 definition to include it. An object is also defined as a context that represents a concept, manages information and provides ways to function on it. Is object-oriented the C language? Not built-in to the language, but by using a framework as the means, object-oriented actions can be emulated. To store data and then store function pointers in the structure. The operation that can be performed on the object . The implementation of a dynamic dispatch table 3 can also take it further.

IV. Goroutines and Channels

Goroutines are advertised as lightweight threads with a small initial stack, having little creation and context switch overhead. The Go runtime multiplexes one or more goroutines within native OS threads as well as moving them between threads. This is done to maximize resource utilization when blocking occurs in goroutines. To make the goroutines cheap they start off with a very small stack segment of 8KiB(4KiB in earlier versions) which is grown on demand. The proceeds. However, this is now considered to be the wrong approach by the Go developers. Consider the amount of work done by creating and freeing segments if a function that allocated a lot of memory is called repeatedly. Because of this problem referred to as “hot split” or “stack trashing”, the next release of Go, 1.3, will have a contiguous stack allocation that grows and shrinks very much the same like a typical dynamic array [32] [33]. Other programming languages projects, like Rust, have come to the same conclusion after experimenting with split stacks [34].

Whole stack is segmented and growth and shrinking is accomplished by simply maintaining linked lists of segments. This makes it cheap to start a routine and easy to both grow and shrink as execution proceeds. However, this is now considered to be the wrong approach by the Go developers. Consider the amount of work done by creating and freeing segments if a function that allocated a lot of memory is called repeatedly. Because of this problem, referred to as “hot split” or “stack trashing”, the next release of Go, 1.3, will have a contiguous stack allocation that grows and shrinks very much the same like a typical dynamic array [32] [33]. Other programming languages projects, like Rust, have come to the same conclusion after experimenting with split stacks [34].

Syntactically a new routine is started with the go statement followed by a function call. Often function call is to an anonymous function which is called a function literal in Go as shown in listing 2.11. These functions are closures meaning that they have access to the variable scope where it was created [20].

Syntactically a new routine is started with the go statement followed by a function call.Often function call is to an anonymous function which is called a function literal in Go as shown in listing 2.11. These functions are closures meaning that they have access to the variable scope where it was created [20].

V. Conclusion

Overall it was really useful to understand the concept of new programming language and to understand why its differ from other languages and so on, for me the interesting features which attracted myself to GOLANG is its concurrency, channels, interfaces, inheritance, unit testing, packages, so overall Go is a language, easy to learn for fresher’s compare to the other experienced developers, because they can quickly understand and adopt. An experienced C# developer, will take some time to understand the changes, if someone is open minded and willing to unlearn some standard programming concepts he learned and practiced in Java or C#.

References

[1] Erik Westrup. Master's thesis work carried out at Axis Communications AB for the Department of Computer Science, Lund University.

[2] Fredrik Pettersson. Master's thesis work carried out at Axis Communications AB for the Department of Computer Science, Lund University.