**GOLANG**

**Basic**

**What is Go?**

GO is an open source programming language which makes it easy to build simple, reliable and efficient software. Programs are constructed from packages, whose properties allow efficient management of dependencies.

**Why should you learn Go?**

* go is easy to learn
* go has an easy to use interface
* great for web development, even front end
* loads of work out there for Golang
* lot of current top-notch applications are being written in Go

Go is good for performance and ease-of-use. Go is a compiled language, giving it a performance which is on-par with native C code. Go provides many of the high-level features which are common in interpreted languages, such as garbage-collection. Go also integrates concurrent programming concepts which a great advantage when programming for multi-threaded, multi-core platforms.

**What is Go suitable for?**

**The really good: Network and Web server:**

* Network applications live and die by concurrency, and Go's native concurrency features -- [goroutines](http://www.infoworld.com/article/2683845/google-go/164121-Fast-guide-to-Go-programming.html" \l "slide7" \t "_blank) and [channels](http://www.infoworld.com/article/2683845/google-go/164121-Fast-guide-to-Go-programming.html#slide8), mainly -- are well suited for such work

**Stand-alone command-line apps or scripts:**

* Another advantage Go has here is speed. The resulting executables run far faster than vanilla Python, or for that matter most any other dynamically executed language

**Explain advantage and disadvantage of Go?**

**Advantage:**

* GO compiles very quickly
* **Golang is open source:** a programming language to be open-source to become better, cleaner, and more efficient
* **Golang is fast:** Go has a simple structure and syntax.  It’s compiled so it provides faster feedback, shorter time to market, and saves time and money.  It’s simple, so it is more maintainable, and development is faster and cheaper.
* **Golang is concurrent:** Golang has efficient concurrency, like C, C++, Java, and at the same time concurrency in Go is done much easier thanks to goroutines, channels, and garbage collection
* **Golang has a garbage collecto:** A form of automatic memory management which has a significant influence on performance and helps to make concurrency more efficient.

**Disadvantages :**

* **No generics support**
* **Err everywhere.** You need to check errors for each of the error producing function in your code explicitly

**How Go is different from the programming languages that you have used? (Java)**

|  |  |  |
| --- | --- | --- |
|  | **GO** | **Java** |
| Architecture | Go does not provide any VM such as Java JVM. This language only compiles to metal like c++/c | It combines both interpretation and compilation approach. Bytecode is interpreted by Java Virtual Machine. Machine code generated by JVM and executed by the system in which Java program runs |
| Language | It is an independent programming language and has at least two compilers such as gccgo and go | Java is an independent language. |
| Expression Syntax | The syntax on Go is specified by the use of extended Backus-Naur Form (EBNF). | Syntax the same everywhere – independent of an IDE or a compiler |
| Mobile Support | The Go mobile subrepository includes mobile support for mobile platforms like iOS and Android and offer tools for building mobile apps. | Depends upon the device manufacturers. |
| Routing | Uses HTTP protocol for routing configuration | Uses Akka.routing.ConsistentHashingRouter and Akka.routing.ScatterGatherFirstCompletedRouter for routing configuration |
| Dependency Injection | Uses dependency injection | Uses dependency injection and allows modification |
| Structure | Easily manageable | Better structure, user- friendly, easier to create and maintain large applications. |
| Speed | Relatively faster than Java | Java is slower than Go |

## Array and Slice

## Array:

## Len and cap always is equal

## Size fixed

## Slice:

## Size can be extended

## Array := [2000]int{1,2,3}

## S1 := append([]int{}, array[:2]…) that gabage collection

## Can’t compare 2 slice

## Byte.Equal()

## Sort.Ints(S)

## Use interface sort

## Map:

## v, ok := m[key]string. Ok is true when key exist

## Not order

## Delete(m,”key”)

1. **What are the differences between slice and link list?**
2. **How re- slicing process of a slice happen under the hood?**
3. **Can you modify array length? Can you append new item into an array?**

* Can’t modify array length

## String

**How is string in Go constructed? And what is the different between string in Go with other programming languages**

* Strings are defined between double quotes "..." and not single quote
* If you use [**range**](https://gobyexample.com/range) within for loop, range will return rune and **byte index** of the character-
* Use ` ` (raw string literal) that down the line often use to in regex and html
* Check is string : str.Validstring()
* Str.Trim()
* Covert string -> Int: strconv.Atoi(str)

1. **What is string literal? How is it done in Go?**

## Goroutine

## Paralleism: follow order

## Fun main() is Goroutine

**What are the difference between goroutine and OS thread?:**

* OS is managed by the operating system

**Explain what is go routine in GO? How you can stop go routine**?

* A goroutine is a lightweight thread of execution: multiple goroutines can be run inside 1 OS thread.
* Key different from other language multithreading.
* Key feature to bring you to concurrency/parallel in Go
* Easily launch goroutine by prefix your method call with keyword go
* A goroutine is a function which is capable of running concurrently with other functions
* To stop goroutine, you pass the goroutine  a signal channel, that signal channel is used to push a value into when you want the goroutine to stop.  The goroutine polls that channel regularly as soon as it detects a signal, it quits

1. **What is a worker pool? And how to do that in Go?**

* A worker pool in my opinion is a grouping of workers (threads, processes, coroutines), that are running the same general task definition, or code reading from a queue of work tasks.
* What I like to think of is a fast food restaurant when thinking about worker pools. Imagine your code is a fast food restaurant. You have a customer walk up to the counter and order a hamburger
* his worker takes the order, runs back to the kitchen, makes the burger, brings the burger back to the customer, and repeats for the next customer. This is such a terrible restaurant that you leave while waiting for hours in line to get your order in, and then you blog about it and the restaurant goes out of business. Failure**.**
* “n” workers take “n” customer orders all at once, so each customer order is handled as they come in, and everyone is happy

1. How to stop a worker pool?
2. What is fan-in, fan-out?
3. What is sync.WaitGroup used for?

- sync.WaitGroup provides a goroutine synchronization mechanism in Golang, and is used for waiting for a collection of goroutines to finish.

1. What are the differences between sync.Mutex and semaphore?

* Using Mutex:

+ A mutex provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work. As long as the buffer is filled by producer, the consumer needs to wait, and vice versa.

* A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that called the wait function

1. **What are the differences between concurrency and parallelism?**

* **Concurrency** is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. For example, multitaskingon a single-core machine.
* **Parallelism** is when tasks literally run at the same time, e.g., on a multicore processor.

1. **What are the differences between sharing by communicate vs communicate by sharing?**
2. **What is channel? How does it work? What is it used for?**

* Go provides a unique concurrency synchronization technique, channel. Channels make goroutines share memory by communicating
* A **channel** is a communication object using which goroutines can communicate with each other. Technically, a channel is a data transfer pipe where data can be **passed into** or **read from**. Hence one goroutine can send data into a channel, while other goroutines can read that data from the same channel.

1. **What is context? What is it used for?**
2. **Have you used goroutine in your application? Describe it.**
3. **Why goroutine is better than thread?**
4. **What is goroutine leak in Go? How to prevent it?**

* A common type of memory leak is leaking Goroutines. If you start a Goroutine that you expect to eventually terminate but it never does then it has leaked. It lives for the lifetime of the application and any memory allocated for the Goroutine can’t be released. This is part of the reasoning behind the advice “Never start a goroutine without knowing how it will stop”.

## Interface

1. **Explain Interface in Go**

In GO, interfaces is a way to specify the behaviour of an object.  An interface is created by using the **“type”**word, followed by a name and the keyword interface.  An interface is specified as two things.

* A set of methods
* Also it is referred as type

1. **How interface in Go different from interface in other languages?**

* Go does not have generic types, protected inheritance, static members, abstract types, final members so those annotations in Java interfaces are unavailable
* Java interfaces can provide a default implementation for methods that an implementing class may choose not to override. Go interfaces cannot.

1. **When should you use interface?**
2. **How did you use interface in your previous project?**

## HTTP

1. Have you ever used any framework for working with HTTP in Go? Compare it with native HTTP library provided by Go.
2. What is middleware? Have you ever written one?

## Performance

1. How many query/second of your application?
2. How can you measure performance of your application?
3. How do you scale up your application?

## Others

1. Does Go pass by value or pass by references? What are the differences?
2. What is Go run time?
3. How does garbage collector work in Go?
4. What is inside the interface{}?
5. How to handle gracefully shutdown in Go?
6. Why would you prefer to use an empty struct{} ? Provide some examples of the good use of the empty struct{}
7. How do you compare two structs? What about two interfaces? Provide examples.
8. What wrongs with following snip of code?
9. How do you structure your Go projects? Explain the advantages and disadvantages?
10. How do you manage configurations of your services in Go?

**Iota:**

* When not have value it will get the last near value

**Panic:**

* **Panic** is a built-in function that stops the ordinary flow of control and begins *panicking*. When the function F calls panic, execution of F stops, any deferred functions in F are executed normally, and then F returns to its caller. To the caller, F then behaves like a call to panic. The process continues up the stack until all functions in the current goroutine have returned, at which point the program crashes. Panics can be initiated by invoking panic directly. They can also be caused by runtime errors, such as out-of-bounds array accesses

**Defer:**

* A **defer statement** pushes a function call onto a list. The list of saved calls is executed in *Last In First Out order*after the surrounding function returns. Defer is commonly used to simplify functions that perform various clean-up actions

**Recover:**

* **Recover** is a built-in function that regains control of a panicking goroutine. Recover is only useful inside deferred functions. During normal execution, a call to recover will return nil and have no other effect. If the current goroutine is panicking, a call to recover will capture the value given to panic and resume normal execution.

**Flask:**

Flask, Django, Tornado, Pyramid

**VueJS**

 4 version khác nhau và cao nhất là version 2.3.4

**PostgreSQL**

- PostgreSQL is a general purpose and object-relational database management system, the most advanced open source database system.

- PostgreSQL is free and open source software.

**Docker**

**GoTest**

go test -v -run=" string name funtion " : test theo ten function

* create a file with a name that ends in \_test.go
* TestXxx: not starting with lowercase letter
* It takes a single parameter that is an instance of \*testing.T