Go Concurrency

Presented by Uditangshu Chakraborty

What I Learnt From this project?

- If/else and switch statements.
- For-Range loops, defer statements.
- Arrays, Slices , Pointers, Interfaces, Map. Function closures
- goroutines, WaitGroup, mutexes and channels, select statements.

01. Read documentation

02. Watching Tutorials

03. Practicing problems on Go.

Starting with the language:-

- Learnt about the package declaration.
- Import statements after the packafge declaration.
- Main function acting as the entry point in the Go code.
- Indentation and formatting is necessary for Go.
- Go promotes explicit Error Handling where is the last return is the type of the error that has accured over a certain function.

Learning about the imports in Go

- math
- fmt
- bufio
- io/ioutil
- os
- http
- json
- image
- strconv
- net

What is concurrency?

Concurrency in Go refers to the ability to execute multiple tasks or processes simultaneously, allowing different parts of a program to run independently and potentially in parallel. In Go, concurrency is primarily achieved through Goroutines and Channels, which are built-in language features designed to simplify concurrent programming and enable the development of efficient and scalable concurrent applications.

Why is concurrent programming different from sequential programming?

- Concurrent programming doesn't follow a **linear flow** whereas sequential programming has a linear flow of control.
- Concurrent programming can offer better performance, scalability, and responsiveness by **leveraging parallelism** and executing multiple tasks **concurrently**.
- Sequential programming may not fully utilize the available resources, while concurrent programming can effectively **utilize multi-core processors** and optimize resource usage through parallelism.

How concurrent programming is done in Go?

Concurrency in go is done through four features, they are:-

- Goroutines
- WaitGroups
- Channels
- Mutexes

1. **Goroutines :-**

- **Goroutines**: Lightweight threads of execution managed by the Go runtime.
- **Creation**: Defined using the **go** keyword followed by a function call or anonymous function.
- **Example**:

```
go func() {
    // Code to execute concurrently
}()
```

2. **WaitGroups :-**

- **WaitGroups**: Synchronization mechanism provided by the **sync** package to wait for a collection of Goroutines to finish executing.
- **Usage**: Add Goroutines to the WaitGroup using **Add()**, signal completion using **Done()**, and wait for all Goroutines to finish using **Wait()**.
- **Example**:

```
var wg sync.WaitGroup
 for i := 0; i < 3; i++ {
 wg.Add(1)
 go func(id int) {
 defer wg.Done() // Decrements the value in wg }(i) }
  wg.Wait()
```

3. **Channels :-**

- **Channels**: Communication mechanism for synchronizing and sharing data between Goroutines.
- **Types**: Unbuffered (synchronous) and Buffered (asynchronous) channels.
- **Operations**: **chan, make(chan type), <-chan, chan<-(unilateral channels)**
- **Example**:

```
ch := make(chan int)  // Unbuffered channel
go func() {
      ch <- 42  // Sending data
}()
value := <-ch  // Receiving data
```

4. Mutexes :-

- **Mutexes**: Mechanism for ensuring exclusive access to shared resources to prevent **deadlock** errors.
- **Usage**: Lock shared data using **Lock()**, unlock it using **Unlock()**, and safely access shared data between locked and unlocked sections.
- **Example**:

```
var mu sync.Mutex
var counter int
go func() {
        mu.Lock()
        counter++
         mu.Unlock() }()
```

How concurrency in Go is different from concurrency in other languages .

- **Goroutines** have smaller memory footprints, faster startup times, and lower overhead, enabling the creation of thousands or even millions of concurrent Goroutines without significant performance degradation.
- **Channels** :-Go provides built-in channels as a first-class communication mechanism for synchronizing and sharing data between Goroutines, promoting a clean and idiomatic approach to concurrent programming.
- Go's **standard library** provides a comprehensive set of concurrency primitives, including Goroutines, channels, WaitGroups, Mutexes.
- Go's memory management and garbage collection system help prevent common memory-related issues like memory leaks, dangling pointers, and buffer overflows.

Github link:-

https://github.com/uditangshu/Go

Thank you very much!