Introduction to Go Programming

Smita Vijayakumar
smita@exotel.in

Agenda

- Modern Compute Environment
- Focus: What Problem does it Solve?
- History
- How is it Achieved
- For C Programmers - Major Differences

Challenges in Modern Compute Landscape

Environment

- Multi-Core
- Networked Systems
- Computational Clusters
- Web Programming Model

Environment

- Thousands of Programmers
- Same code base
- Scale of Development

Legacy Languages

- Language Feature Complex
- Garbage Collection
- Missing language support for Concurrency
- Compilation Times
- Dependency Management

Crux:

Programmers digress from the real task!

Focus

“Go’s purpose is therefore not to do research into programming language design; it is to improve the working environment for its designers and their coworkers.”

–Rob Pike (Co-Designer of Go and Distinguished Engineer at Google)

History

History

- Inception: 2007
- Go 1.0 released in 2012
- Rob Pike, Ken Thompson and Robert Griesemer

History

- Binary = 2000 X C++ Source Bytes
- Unused Library Includes
- Conditional Includes
- Extremely Slow Compilation
  - Even on Distributed Compilation System!

Desired Features

- Light Weight
- Increase Programmer Efficiency
- Compiled Language
- Concurrency
- Garbage Collection
- Statically Typed

How is it Achieved?

How

- No unused import packages
- Dependency graph is precise
- No dependency cycles
- Only exported data available
- One object both exported and complete data

How

40X faster compilation times than C++!

Differences between Go and C

Syntax

- Import paths are URLs
- Cleaner syntax

Syntax

```
//C
typedef struct A {
    int a;
    char b;
}

//Go
type A struct {
    a int
    b char
}
```

Syntax

```
/* C */
//Expression Syntax
int main(int argc, char *argv[])

// The Clockwise-Spiral Rule in C syntax parsing
int (*(*fp)(int (*)(int, int), int))(int, int)

/* Go */
// Type Syntax
func main(argc int, argv []string) int

//Left to right
f func(func(int,int) int, int) func(int, int) int
```

Syntax

```
//Go can also return multiple value from function
func ReadFile(r io.Reader) (num int, err error) {
…
}
n, err := ReadFile(r)
```

Syntax

```
package util

//Counter is visible when package is imported
var Counter int

//name is not
var name string

//Seen
func ThisIsAlsoAvailable() error {
}

//not seen
func thisIsAPrivateFunction() error {
}
```

Scopes

- Universe (language identifiers)
- Package
- File
- Function
- Block

Semantics

- No pointer arithmetics
- No implicit numeric conversions
- Array bounds are always checked
- types are not type aliases
  - type X int // X and int are distinct
- Interface types
- Reflection
- Type switch

Concurrency

- Go Routines
- Channels - Communication Pipes

Concurrency

Go Routines

- Light-weight threads of execution
- Managed by Go Runtime
- Run concurrently with other go routines
- Example
  - Shared resource manager
  - Querier
  - DB Reader
  - etc…

Concurrency

```
package main
import “fmt”

func myPrinter(str string) {
    fmt.Println(str)
}

func main() {
    myPrinter(“one”)
    go myPrinter(“maybetwo”)
    go myPrinter(“maybethree”)
}

//Output
one
maybetwo
maybethree

//Or
one
maybethree
maybetwo
```

Concurrency

Channels

Shared pipes connecting concurrent go routines

Concurrency

```
package main

import "fmt"

func main() {
    messages := make(chan string)

    // Send a value into a channel using the `channel <-`
    go func() {
        messages <- "ping"
    }()

    // Receive a value from channel using the `<-channel`
    msg := <-messages
    fmt.Println(msg)
}
//Output
ping
```

GC

- Easier for Programmer
- No overhead of memory allocation and freeing
- No explicit memory freeing

Interfaces

- Set of methods
- Methods on any type
- Data types implementing all methods satisfy