Introduction to Go Programming for Developers

Course Summary

Description

The Go programming language (or golang), created at Google by Rob Pike and Ken Thompson, is one of the prominent new programming languages that has been developed to meet the needs of large scale application development, modern development methodologies like Agile and DevOps, high performance multi-core hardware technology and the need for increasing scalability. Go was designed to replace legacy programming languages in current use which were not designed for the massive scale of integration and the shrinking release cycles required by software projects in modern tech organizations like Google.

Go is a C-style language which is syntactical similar to other C-style languages like C and Java, however Go contains a number of innovations to enable programmers to write high performance code that is still execution safe. Concurrency and other core language features allow Go applications to exploit the power and speed of modern high performance technical architectures. Go is also a complete development, build and test environment designed to support rapid delivery methodologies where large numbers of programmers can collaborate on highly scalable, multiple component applications with rapid build and short delivery cycles. The ability to integrate Go packages from anywhere on the Internet into a build, similar way to R, Python and Ruby, allow highly distributed teams to work in real time without any reliance of third party tool chains.

The course starts with an overview of the basic language structure from data types, functions, packages, libraries, pointers and the development environment exploring the unique features of Go and how these differ from other programming languages while emphasizing Go best practices for using those features. This is followed by a discussion the built-in data structures of Go such as arrays, maps and slices but with an emphasis again on how to use these Go constructs to write high performance code. Also covered is the unique Go error system and the use of deferred functions to create high tolerance code.

The use of structs, interfaces and methods are explained in detail emphasizing again the Go best practices, how to use these program constructs to create user defined types and how to use these to implement object oriented style program designs. However the course also covers the use of Go to implement func-tional programming designs using functions as first class objects, anonymous functions, encapsulations and other related functional programming techniques.

Integrated into the course is a tour of the Go build system, vendor management, Go libraries, profiling and benchmarking utilities as well as the unit test facilities for test driven development. The course also explores in some depth how concurrency works in Go using goroutines and channels, including how to design code to take advantage of concurrency in order to produce high performance applications. This includes a brief introduction of Go concurrency patterns and architectures.

The class is designed to be about 50% hands on labs and exercises, about 25% theory and 25% instructor led hands on learning where students code along with the instructor.

The course “Advanced Go Programming for Developers” (PT20183) is designed to follow this course.

Introduction to Go Programming for Developers

Course Summary (cont'd)

Topics

- Go syntax – variables, data types, functions, packages, and errors.
- Go data structures – arrays, slices and maps.
- User defined types – structs, embedded structs and interfaces.
- Methods and interfaces – writing OO applications with types, methods and interfaces.
- Survey of the the Go packages.
- Using the Go tool chain for continuous integration, test and delivery.
- Testing, benchmarking, tracing and profiling in Go.
- Large scale project dependency management using the Go tool chain.
- First class functions, encapsulation and functional programming in Go.
- Go concurrency – goroutines, channels, and concurrency designs.
- Best practices and the Go specific programming idioms and patterns.

Audience

This course is designed for experienced developers in either a C-style language like Java, C, C++, C#, JavaScript or similar language.

Prerequisites

Before taking this course, students should have at least an intermediate level or experience in an object oriented language like Python or Ruby. Due to the pace of the course, it is not recommended for beginning or inexperienced programmers.

Duration

Three days

Introduction to Go Programming for Developers

Course Outline

I. **Introduction to Go**
   A. The problem: legacy tools in modern IT environments.
   B. Design philosophy and objectives of Go.
   C. “Hello World!” Go program structure and format.
   D. Features of Go that differentiate it from other languages.
   E. Features for high performance builds.
   F. Features to support modern development processes (e.g. DevOps).
   G. The Go tool chain introduction: run, build, install and format.
   H. Go application structure, packages, importing and dependencies.
   I. Where Go does things differently and why.

II. **Basic Go Syntax**
   A. Variables and data types.
   B. Full and inline variable assignment.
   C. Package visibility and variable scoping.
   D. Using pointers in Go.
   E. Runes, strings and characters.
   F. Constants and iota.

III. **Go control structures**
   A. Conditionals.
   B. In-line variable declarations.
   C. The for loop.
   D. The range() function.
   E. The switch() statement.

IV. **Function Basics**
   A. Named and naked returns.
   B. Multiple return values.
   C. Varadic functions.
   D. Recursion.
   E. The “defer” operator.
   F. Generating and returning errors.
   G. Handing errors and panics – the “comma ok” idiom.

V. **Data Structures**
   A. Arrays.
   B. Maps and hashtables.
   C. Slices – creating from arrays and standalone slices.
   D. Efficient use of maps, slices and arrays with functions.
   E. Array functions and operations.
   F. Map functions and operations.
   G. The make() function.

VI. **User defined types**
   A. User defined types: aliases and structs.
   B. Defining, initializing and accessing structs.
   C. Using struct pointers.
   D. Using the new() function.
   E. Embedding structs,
   F. Anonymous fields,

VII. **Methods**
   A. Methods and receivers.
   B. Method sets.
   C. Methods and embedded structs.
   D. OO programming using structs and methods.

VIII. **Interfaces**
   A. Defining and using Go interfaces.
   B. The empty interface {}.
   C. Interface variables.
   D. Type testing and the switch() statement.

IX. **Concurrency**
   A. The Go concurrency model.
   B. Goroutines.
   C. Channels, directionality, type, buffered and unbuffered.
   D. Multiplexing with the select() statement.
   E. Synchronization of goroutines by using channels.
   F. Concurrency patterns and best practices.

Introduction to Go Programming for Developers

Course Outline (cont'd)

G. Designing for concurrency.
H. Race conditions.

X. Testing, Benchmarking and Profiling
A. The Go test tool.
B. Unit testing and TDD in Go.
C. Benchmarking in Go.
D. Go profiling.

XI. Go Packaging
A. Workspace organization.
B. Using "go get".
C. Package initialization.
D. Vendor management.
E. Supporting continuous delivery.
F. Platform specific releases.
G. Team support for a shared code base.

XII. Functional Programming with Go
A. Functions as first class objects.
B. Anonymous functions.
C. Encapsulations.
D. Implementing functional programming in Go.

XIII. Go Libraries
A. The fmt package.
B. The io and os packages.
C. The http net package and json package.
D. The string and sort packages.
E. The sync package.
F. The log, time and flag packages.
G. The cgo (linking with C) package.

XIV. The Golang Way of Coding
A. Go application design.
B. Go idioms and patterns.
C. Go best practices.
D. Efficient Go code.
E. Rookie mistakes with Go to avoid.