Project 0: Golang Intro & WhatsUp

Due: 10:29 AM, Jan 26, 2017

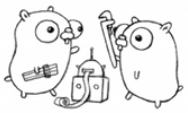
Contents

1 Introduction

Welcome to CS 138! This project is a self-contained introduction to the course and should help you familiarize yourself with the Go programming language. You'll be implementing a small chat application (including two components: client & server).

We don't assume that you have experience with Go, therefore most of this assignment is targeted at getting you comfortable with using Go, which will be used in all of the projects in this course.

2 The Go Programming Language



Go is an open-source programming language created by a team at Google (and other outside contributors). Go was initially started in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson. Go is a systems language with roots in C, C++, and other languages. Version 1 of Go was released in 2012 and is under active development (v1.7 was released 8/2016). If you have more questions about Go's history there's a wonderful FAQ[1] on their website which we urge you to checkout.

2.1 Installing Go

The department machines have Go available as a contrib project (v1.7) at `/contrib/bin/go`, but in case you want to use your own machine we urge you to follow the official installation directions[2] for more details. If you are on a Mac and use Homebrew[3], `brew install go` will install the latest version of Go (v1.7).

2.2 A Tour of Go

"A Tour of Go"[4] is an interactive tutorial that helps teach many of the interesting bits about Go. Before we start our chat application (WhatsUp) we highly recommend you that you go through this tour (even if you have previous Go experience you may learn something new).

There are 12 exercises spread throughout the tour. We recommend you do most of them since they are useful in making sure you understand how things are done in Go. We are not requiring you to turn in anything from this part of the assignment.

2.3 Writing Go Code

Once you finish the tour, read "How to Write Go Code"[5] to understand how to organize code, write a package or library, test your code, and work with the command line tools. Make sure to setup your `$GOPATH` appropriately and become familiar with the Go command-line tool. The article is also available as a screencast[6].

In addition, skim through Effective Go[7], and the Go FAQ[8]. Both are great reference documents that you should come back to from time to time.

3 WhatsUp

WhatsUp is a small chat application that somewhat resembles an application with a similarly sounding name. WhatsUp supports sending and receiving messages in a user-to-user manner (like Google Hangouts) instead of broadcast (like IRC or group chats). This means that if three users

[1] `https://golang/doc/faq`
[2] `https://golang.org/doc/install`
[3] `http://brew.sh/`
[4] `https://tour.golang.org/`
[5] `http://golang.org/doc/code.html`
[6] `https://www.youtube.com/watch?v=XCsL89YtqCs`
[7] `https://golang.org/doc/effective_go.html`
[8] `https://golang.org/doc/faq`

(Tom, Rodrigo, and Ugur) are connected to the same server, Tom can send a message to Rodrigo that will not be delivered to Ugur. The server is centralized (for simplicity) and has the responsibility of routing messages to and from the connected users of the system.

3.1 Getting Started

You can get the stencil code by running `go get github.com/brown-csci1380/whatsup`. This should clone the repo into `$GOPATH/src/github.com/brown-csci1380/whatsup`. You can edit your code, and run `go build` to compile in-place, or `go install` to compile and install the `whatsup` binary into `$GOPATH/bin`, allowing you to run it directly.

This step is new this year, so if it is holding you back, feel free to also download a copy of the stencil code from the website and work within that directory.

Within the stencil, we provide a small amount of library code to help standardize the communication between a client and server. You are provided a protocol library (`whatsup/proto.go`) so that everyone uses the same format for sending/receiving messages and a TCP listener library (`whatsup/listener.go`) so that everyone uses unique random listening ports. Since everyone will be using the same protocol you can use it to chat with your friends.

3.2 Protocol

The `ChatMsg` struct along with the `Purpose` enum contained in `whatsup/proto.go` define the messages that are exchanged between clients and a server. All messages between a client and server should use gob encoding[9] to send WhatsUpMsg structs. The support code makes use of these packages to serialize and deserialize messages for you.

3.3 WhatsUpMsg Struct

- A message contains three data parts: `Username string, Body string, Action Purpose`.
- The type of message and the meaning of the fields is defined by its `Purpose`, which can be any of the following: `CONNECT`, `MSG`, `LIST`, `ERROR`, `DISCONNECT`.

3.4 Message Semantics

- `CONNECT`:
  - Description: upon starting a client it sends a message of this type to initiate a connection to the server for future message transport.
  - `WhatsUpMsg.Username` contains the username of the client attempting to connect with the server.
  - `WhatsUpMsg.Body` is not defined for this type of message.

[9] `http://golang.org/pkg/encoding/gob/`

```go
// Enum
type Purpose int

const (
    CONNECT Purpose = 1 + iota
    MSG
    LIST
    ERROR
    DISCONNECT
)

type WhatsUpMsg struct {
    Username string
    Body     string
    Action   Purpose
}
```

Figure 1: structs from `whatsup/protocol.go`

- `MSG`:
  - Description: a chat msg sent from one a client to another user (may be itself).
  - `WhatsUpMsg.Username` is the username that the msg is destined for (e.g. Tom).
  - `WhatsUpMsg.Body` contains the actual msg (e.g. hello world).
- `LIST`:
  - Description: a client can ask the server what users are currently connected
  - `WhatsUpMsg.Username` is not defined for this type of message.
  - `WhatsUpMsg.Body` is empty if this is a request and contains a list of connected users if it is a response.
- `ERROR`:
  - Description: if the server cannot satisfy a request for whatever reason it should send an error message back to the client saying why it cannot.
  - `WhatsUpMsg.Username` is not defined for this type of message.
  - `WhatsUpMsg.Body` contains the error string
- `DISCONNECT`:
  - Description: a client can ask to be disconnected gracefully from the server
  - `WhatsUpMsg.Username` is not defined for this type of message.
  - `WhatsUpMsg.Body` is not defined for this type of message.

3.5 Server

The server is responsible for conforming to the protocol detailed in the previous section. It's primary responsibility is to route messages between client users. You must use Go routines and may use Go channels to deal with concurrency. Remember that in concurrent programming you must find a way for data to be accessed safely with no race conditions. When programming with Go routines and channels you may find places where mutexes will come in handy.

In order for the server to start it needs to listen for incoming connections on a defined port. We've taken care of this part for you in the listener library (`whatsup/listener.go`), use the function `OpenListener() (net.Listener, int, error)` to do this. It returns a Go `net.Listener` object, an integer that is the port it successfully listened on and an error (this is nil if there were no errors). The remainder of the server design and structure is left up to you.

3.6 Client

Upon starting up, the WhatsUp client should connect to a user defined WhatsUp server. From there it should take user input (e.g. via a command prompt) to send and receive messages between it and the server. We leave the overall design of this portion to your best judgment.

3.7 Testing

We will not be fully testing your code (unlike future projects!), but we do expect you to have at least a few tests (i.e., > 1) so that you have a chance to get familiar with how testing in Go works. Please refer to the testing documentation[10] for examples on how to setup Go tests for your project.

4 Handing In

We will not be grading this assignment in detail (unlike future projects!). This project is more geared towards helping you get a head start on Go so that future projects will be a bit easier. The directory structure of your assignment should include the following:

- README
  - Please write up a simple README documenting any bugs you know of in your code, any extra features you added, and anything else you think the TAs should know about your project.
- Chat
  - The contents of your project and all required source code to build and run it.

If you have pre-registered for the course, you should run the electronic handin script

```
/course/cs138/bin/cs138_handin whatsup
```

[10] `http://golang.org/pkg/testing/`

to deliver us a copy of your code.

If you are not pre-registered for the course, you should email the TA staff your handin by the deadline.

5 Grading

Unlike future projects this project will not count towards your final grade. However, we will evaluate your submission and provide feedback. The main goal of this project is for you to get familiar with Go! Your program should successfully compile and pass all the tests you provide. Your code should also be able to interoperate with our implementation and with other students' implementations, as they will all use the same message formats and semantics.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS138 document by filling out the anonymous feedback form:

`http://cs.brown.edu/courses/cs138/s17/feedback.html`.