The Go Programming Language

Jinuk Kim (rein@ncsoft.com)
Server Platform Team, Studio 7, NCSOFT

2010. 06. 23

Not-yet covered Go Constructs

- ‘defer’ construct for deferred (clean-up) execution
- Variadic function
- Iteration using range( ) on containers, strings, channels
- C binding
- Pitfalls of goroutines
- Package

Deferred Execution

- Go has “defer” construct for cleanup

```
func blah(...) {
    defer cleanUpFunction( ... ) // or any expression
    /* other function calls */
}
```

- ‘cleanUpFunction’ will be called at **stack-rewinding** (scope-rule)
- Similar to *with* construct of python programming language

Variadic Functions

- Function can take variable number of arguments, as in C/C++
- eg) **func** variadicFunc(arg ... **interface**{ });
- In above example, the most generic “**interface** { }” used to handle any kind of argument type
- Or, types can be specified: “**func** variadicStrFun(arg ... string);”
- arguments can be referred using range( ) construct

range( ) construct: over container

- Iterates over built in container, array, slice (*of array*), map
- Uses *pythonic* syntax, iterating over (key, value) pair

  -
    ```
    a := make([ ]int, 8) // a := [0, 0, ..., 0]
    for k, v := range a { // gives (0, 0), (1, 0), ... , (7, 0)
        fmt.Println(k, v)
        v = k * k
    }
    fmt.Println(a) // displays [0, ..., 0] ; (k, v) pairs are not references
    ```

range( ) over built-in string

- Built-in string contains UTF-8 encoded text
- It’s immutable & byte-indexed
- Built-in len( ) function gives byte-length, so for per-character operation
- Special “**for, range**” clause for string literals, (or string variables)
  - **for** pos, char := **range**(“한글”) { *// for array, it was (index, value) pair*

    fmt.Printf(“%c(%d) ”, char, pos)

    } *// displays “한(0) 글(3)”*

range( ) over ‘go channel’

- Iterate over channel, as if channel is a container

    -
      ```
      func sinker(ch chan int) {
          for _ = range(ch) {
              ...
          }
      }
      ```

range( ) over variadic argument list

- Variadic argument list is just a slice of specified type T
  - **func** variadicFunc(args ... **interface** { }) {
        for arg := **range**(args) {
            ... *// arg can be any type*
  - *cf)* **func** variadicStrFunc(args ... string) {
        // args is just a splice of string. ([ ] string)

Pitfalls of goroutines

- Compare following go code and C++ code

  -
    ```
    x := 0
    go func( ) { x++; fmt.Println(x); } // prints 1 or 2, depending on
    go func( ) { x++; fmt.Println(x); } // the execution-order
    ```

  -
    ```
    int x = 0
    [=]( ) { x++; cout << x << endl; } ( ); // prints 0
    [&]( ) { x++; cout << x << endl; } ( ); // prints 1, and updates x
    ```

Pitfalls of goroutines

- goroutines refers *up-value* **by reference**, which can cause race-conditions
    - C++ controls *“how up-values are referenced”* in lambda functions
- Should do,
    - Pass it as parameter to goroutine (no-sharing; privatization)
    - Use mutex on shared variable or other synch. primitives

Pitfalls of goroutines

- goroutines run simultaneously up to *GOMAXPROCS*
  - Default value is **“1”**
  - Update it using “runtime.GOMAXPROCS(desired-number)”
- gccgo runs “each goroutine” on independent pthreads.
  - Huge stack size - hit the wall with small # of goroutines
  - Performance penalty - context switching overheads

Bind C Functions from Go

- From trivial SQLite3 binder (*http://code.google.com/p/gosqlite*)
- import “C”: ***pseudo***-package using cgo compiler(with gcc)
- structures: struct sqlite3, struct sqlite3_stmt
  - imported as C.sqlite3, C.sqlite3_stmt
- functions: sqlite3_open, sqlite3_step, ...
  - imported as C.sqlite3_open, C.sqlite3_step, ...

Bind C Functions from Go

- *NULL*-terminated C style string literals
  - create: *str := C.CString( )*
  - destroy: *C.free(unsafe.Pointer(str))*
- Manually convert some argument type on library function calls
  - CString <-> *C.char, C.int
- Get data from C using unsafe.Pointer

User-defined Package

- **package** construct in go-source defines the package
- packages are imported from $(GOROOT)/$(GOARCH)/pkg/**path/to/pakage**. for example,
  - gosqlite package resides in pkg/gosqlite.googlecode.com/gosqlite.*
  - **import** "gosqlite.googlecode.com/gosqlite" imports SQLite binding
- imported namespace can be altered using
  - **import** newName "path/to/package"

User-defined Package

- To be seen outside the package, first letter of the symbol should be **upper-case**
- Package shall use the build-script in $(GOROOT)/src/make.pkg
  - To use library written in C, exploit cgo command line tool
- Imported packages are initialized (using init()) before importing pakage
- Multiply imported package initializes only once

Simple Web Server Implementation

- Utilize multiple goroutines
- Using built-in module http
- Bind handler to URI
  - http.Handle(“/hello”, http.HandleFunc(Hello)) *// register handler*
    http.ListenAndServe(“:80”, **nil**) *// serve forever*
  - **func** Hello(conn *http.Conn, req *http.Request) {
        io.WriteString(conn, “Hello, world”) }

Simple Web Server Implementation

- Able to utilize the multiple cores (with multiple goroutines)
- Similar to Python's Handler of BasicHTTPServer, it serves each request from client
- Can separate the I/O threads from worker threads using go channel
- Can adapt many C-based modules
  - can be compiled into Go package using cgo tool

Q & A

Appendix: Implementation Details

Server Structure

- Manages bookmarks (name, uri pair)
- Add/Remove/List the bookmark(s)
- Match some server-uri to specific role (and http handler)
  - Add bookmark on “/add”
  - Remove bookmark on “/remove”
  - List bookmark(s) on “/list”

Database Backend

- **import** db “gosqlie.googlecode.com/sqlite” *// import sqlite as db*
- **var** dbCon *db.Conn = **nil**
  dbCon, err := db.Open(“test.db”) *// create db connection*
  defer dbCon.Close( ) *// schedule clean-up at exit*
- */* register the handlers */*
- http.Handle(“/list”, http.HandlerFunc(LinkList))
- http.ListenAndServe(“:80”, **nil**)

Handler Implementations (1/3)

```
// Adds bookmark to DB, and redirects to the listing page
func LinkAdd(conn *http.Conn, req *http.Request) {
    req.ParseForm()
    name, uri := req.FormValue("name"), req.FormValue(“uri”)
    dbCon.Exec(fmt.Sprintf("insert into links values('%s', '%s')",
        name, uri))
    http.Redirect(conn, "/list", 303) // HTTP 303 redirection
}
```

Handler Implementations (2/3)

```
// removes specified bookmark, and redirects to the listing page
func LinkRemove(conn *http.Conn, req *http.Request) {
    req.ParseForm( )
    name := req.FormValue("name")
    dbCon.Exec(fmt.Sprintf("delete from links where name='%s'",
        name))
    http.Redirect(conn, "/list", 303)
}
```

Handler Implementations (3/3)

```
func LinkList(conn *http.Conn, req *http.Request) {
    stmt, err := dbCon.Prepare("select * from links")
    defer stmt.Finalize()
    err = stmt.Exec()
    if err == nil {
        link, uri := "", ""
        for stmt.Next() {
            stmt.Scan(&link, &uri);  writeLinkItem(conn, link, uri) ;
}}}
```