RX-M / go Private

Branch: master go / modules / error-handling / rx-m-go-lab-error-handling.md Find file Copy path

ronaldpetty for some reason pdf image was broken, using local versus online 30dea52 on Sep 12

3 contributors

112 lines (72 sloc) 2.72 KB

Go

Error Handling

In this lab, we will explore how Go handles errors.

1. Defer

Defer is used to ensure that some action is performed later in a program's execution. This is usually for cleanup/teardown purposes. Make a new directory called error-handling to house your work for this lab.

```
user@ubuntu:~/go/src/lab-methods-interfaces$ cd
```

```
user@ubuntu:~$


user@ubuntu:~$ mkdir ~/go/src/error-handling

user@ubuntu:~$


user@ubuntu:~$ cd !$

cd ~/go/src/error-handling
user@ubuntu:~/go/src/error-handling$
```

Next create a file called 'error.go' and, using the os package, write a program that creates a file, writes to it, and then closes it. What part of the program should be deferred?

2. Panic and Recover

Take a look at the following code. What is it doing?

```
user@ubuntu:~/go/src/error-handling$ vim defer.go
user@ubuntu:~/go/src/error-handling$ cat defer.go

package main

import "fmt"

func main() {
    f()
    fmt.Println("Returned normally from f.")
}

func f() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("Recovered in f", r)
```

```
            }
        }()
        fmt.Println("Calling g.")
        g(0)
        fmt.Println("Returned normally from g.")
    }

    func g(i int) {
        if i > 3 {
            fmt.Println("Panicking!")
            panic(fmt.Sprintf("%v", i))
        }
        defer fmt.Println("Defer in g", i)
        fmt.Println("Printing in g", i)
        g(i + 1)
    }
    user@ubuntu:~/go/src/error-handling$
```

Enter and run the program to test its function.

3. Error Challenge Step

`error` is a built-in interface type that is either nil (implies success) or non-nil (implies failure). The Go philosophy is that exceptions entangle the description of an error with the control flow required to handle it, so Go programs use ordinary control-flow mechanisms to respond to errors.

Imagine that we need to solve quadrtic equations in our code. Equations like: `ax^2 + bx + c = 0`, where x represents an unknown, and a, b, and c represent known numbers such that a is not equal to 0 are quadratic.

The quadratic formula is an algebraic solution of the quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Write a function with the signature: `quadratic(a float64, b float64, c float64)`

Have this function implement the quadratic formula, accounting for any potential errors (like dividing by zero, taking the square root of a negative number). The function should return two floats (the + and - sides of the square root term) and an error - which would be nil if no error occurs.

Congratulations you have completed the lab!!