Anthony Quintero-Quiroga

Mark Bates

Intro to Go

October 3, 2021

Go Fmt Package

I came across a couple surprises when researching the Go fmt package. One surprise was the number of formatting verbs supported by Go. Compared to other languages, Go has a large selection of formatting verbs. Surprisingly, Go also has formatting verbs for stucts, arrays, slices, maps and pointers. Go fmt package also offers a larger variety of usable functions when compared to other languages. Errorf, Sprintf and Sscanf are functions not seen in other languages outside of C and C#. Java and Python would have to utilize other classes and functions to achieve similar functionality.

The default formatting of printing objects in Go seems to be more useful to the user when compared to other languages, like Java, where the object’s ToString() method is called. From my experience an object’s ToString() method usually needs to be manually overridden to provide any meaningful information to the user. Go’s fmt error messages are surprisingly meaningful as well. The built-in formatting errors along with “go vet” provide the developer exact information on what the problem is and how to fix it. If you have the wrong type or unknown verb like in “Printf(“%d”, “hi”), Go will provide you with the following format error, “%!d(string=hi)”. Similarly, if too many or too few arguments are provided, Go will provide other meaningful formatting errors. Other languages typically just provide a line number of where the formatting error without any additional information.

Another surprising aspect of Go’s fmt package is that print and fprint return the number of bytes written and any write error encountered by default. In a single line of code, a developer can print a statement while storing the bytes written and any error message to separate variables. For example, n, err := fmt.Fprintf(os.Stdout, "%s is %d years old.\n", “Kim”, 21) will store the number of bytes onto variable “n”, the error message in “err”, while print “Kim is 21 years old.” Other languages, such as Java, would require more lines of code to accomplish the same results. This goes to show that Go really tries to be as simple and efficient as possible. The different format types in Go also add some simplicity and practicality when printing structs and objects. Using types such as GoStringer or Stringer allows us to get more meaningful results when printing structs and objects. When printing structs in other languages, we would get hexadecimals and addresses to pointers instead of the actual values of what is stored. For example, fmt.Printf("%#v\n", p2) would have an output of “Person{Name:"Theia", Age:0x4, Addr:(\*main.Address)(nil)}” if GoString() was not implemented. With GoString(), however, we get an output of “Person{Name: "Theia", Age: 4}.”

Overall, Go’s fmt package is surprisingly comprehensive and provides a wide range of utility and features. It almost seems like everything one would need from a fmt package is offered straight out-of-the box. Other languages, like Java, can accomplish the same features but requires the use of additional packages or the creation of custom classes to achieve some of the same functionality. The Go develops really did their research and took the best from competing languages and/or improved upon features those other languages offered. I look forward to seeing what other exciting features Go has to offer and other differences it has compared to other languages.