Tools and Go's Performance

This lesson discusses the tools that make Go outshine other languages performance-wise.

WE'LL COVER THE FOLLOWING

Other tools

The tools in the Go-toolset are partly shell-scripts; some are written in Go itself. They are implemented as commands for the go tool. Here, we discuss only a few of them. For more info, consult the docs.

`go get` tool

It is the go-package manager tool, much like rubygems for the Ruby language. It is meant for installing go packages outside of the standard library, and it works with the source-code format of the package, which then has to be compiled and linked locally.

`go vet` tool

It is a tool to report likely mistakes in the code. This command is followed by the name (or names) of the package(s) that have to be examined. It checks, for example, for useless assignments, consistency in the use of the `Printf` function, unused function results, and so on. Not all problems reported are

genuine, but it can find errors which the compilers did not catch.

`go tool cover` tool

It is a tool that enables coverage analysis. It annotates the source code before compiling, indicating which parts of your code are executed and called. It can be used to detect useless code, or on the contrary, to track which parts are heavily used.

`go fix` tool

It is a tool that you can use to update Go source-code (outside of the standard packages) from an older to the newest release: it tries to automatically *fix* changes. It is meant to reduce the amount of effort it takes to update existing code, automating it as much as possible. The `gofix` takes care of the easy, repetitive, and tedious changes. If a change in API isn't simply a substitution of a new function and requires manual examination, `gofix` prints a warning giving the file name and line number, so that a developer can examine and rewrite the code. The Go-team regularly updates the tool together with new API changes, and it is also used internally in Google to update the Go source code. The `gofix` works because Go has support in its standard libraries for parsing Go source files into (abstract) syntax trees (AST's) and also for printing those syntax trees back to Go source code. `go fix .` tries to update all **.go-files** in the current directory when necessary. The changed file names are printed on standard output. For example:

```
go fix -diff .
```

shows the differences with the installed Go-release. To update with new changes, use:

```
 go fix -w .
```

`go test` tool

It is a lightweight test framework for unit-testing.

Go's performance

According to the Go-team and measured in simple algorithmic programs,

performance is typically 10-20% slower than C. There are no official benchmarks, but regular experimental comparisons between languages show a very good performance track. A more realistic statement is that **Go is 20% slower than C++**.

In some cases, that difference is irrelevant, but for a company like Google with thousands of servers, the potential efficiency improvements are certainly worth the investment. The current popular languages execute in a virtual machine: JVM for Java and Scala, .NET CLR for C#, and so on. Even with the massive improvements in virtual machines, JIT-compilers, and scripting language interpreters (Ruby, Python, Perl, JavaScript), none can come close to C and C++ for performance.

If Go is 20% slower than C++, that's still 2 to 10 times faster than any language that's not statically typed and compiled, and it is far more memory efficient. Comparing benchmarks of a program in 2 or more languages is very tricky: the programs each should do exactly the same things, utilizing the best possible concepts and techniques for that task from the language. For example, when processing text, the language that processes this as raw bytes will almost certainly outperform the language that works with it as Unicode. The person performing the benchmarks often writes both programs, but often he/she is much more experienced in one language than in the other, and this can very much influence the results. Each program should be written by a developer who is well versed in the language. Otherwise, it is not difficult to artificially influence the performance behavior of one language compared to another; it is not an exact science. The outcome can also depend upon the problem to solve. In most cases, older languages have optimally tuned libraries for certain tasks.

Summary

Go has a performance similar to Java and C++, and about **25x** better than dynamic languages like Nodejs or Python. Furthermore, Go has a small runtime compared to Java or .NET, and it also has a much lower memory consumption.

Go has made its place in the programming world. Due to its popularity, its interaction with C and C++ is possible which we'll look at in the next lesson.