pygo: Interpreters in Go and for Go

Sébastien Binet

CNRS/IN2P3

2016-10-10

Go in Science

- efficient and quick edit/compile/run development cycle
- fast at runtime
- robust scientific libraries (*e.g.* `gonum/...`)
- easy deployment
- simple language that scientists can quickly master

But `Go` could use a real interpreter and its `Read-Eval-Print Loop` (`REPL`).
`REPL`s are **fantastic** for science and **exploratory** work.

Go REPL

There are many Go interpreters and REPLs:

- motemen/gore
- sbinet/go-eval
- golang.org/x/tools/ssa/interp
- ...

but none of them provide an interpreter + a REPL for the **full** Go language.

go-interpreter

`github.com/go-interpreter` is a (nascent) community effort to:

- design,
- implement and
- provide an interpreter (+ a `REPL`)

for `Go`, in `Go`.

We are still at the design phase and working on `proposal-1`.

go-interpreter - issue

Main problem (for me, at least): next to **NO** expertise in designing and building interpreters and REPLs:

- components of an interpreter? of a `REPL`?
- use a virtual machine (VM)?
- stack-based or register-based VM?
- opcodes?
- bytecode format? `LLVM`, `WebAssembly`, `dis` or roll our own?

AOSA book

Discovered `"A Python Interpreter Written in Python"` by Allison Kaptur, in `The Architecture of Open Source Applications` book.

Great: let's do that in `Go`, for `(C)Python3`.

Having a blueprint and a much more constrained design space surely will help the learning process of the basic concepts!

pygo

pygo is a (toy) virtual machine interpreter for `CPython`.
A VM for `CPython3`, in `Go`.

Like in the AOSA book:

- use `/usr/bin/python3` to compile source code into bytecode
- feed this bytecode to a VM that will, somehow
- read, decode and **interpret** instructions from the bytecode

```
shell> python3 -m compileall -l my-file.py
shell> pygo ./__pycache__/my-file.cpython-35.pyc
```

Example

Let's say we want to execute the following python script:

```python
a = 1
b = 2
print(a+b)
```

- load a value
- store a value
- add 2 values
- print resulting value

python3 bytecode looks like this:

```
[100, 1, 0, 125, 0, 0,
 100, 2, 0, 125, 1, 0,
 116, 0, 0,
 124, 0, 0, 124, 1, 0,
 23,
 131, 1, 0]
```

Example - pygo/main

```go
package main

func main() {
        code := Code{
                Prog: []Instruction{
                        OpLoadValue, 0,
                        OpStoreName, 0,
                        OpLoadValue, 1,
                        OpStoreName, 1,
                        OpLoadName, 0,
                        OpLoadName, 1,
                        OpAdd,
                        OpPrint,
                },
                Numbers: []int{1, 2},
                Names:   []string{"a", "b"},
        }

        interp := New()
        interp.Run(code)
}
```

```go
func (interp *Interpreter) Run(code Code) {
        prog := code.Prog
        for pc := 0; pc < len(prog); pc++ {
                op := prog[pc].(Opcode)
                switch op {
                case OpLoadValue:
                        pc++
                        val := code.Numbers[prog[pc].(int)]
                        interp.stack.push(val)
                case OpAdd:
                        lhs := interp.stack.pop()
                        rhs := interp.stack.pop()
                        sum := lhs + rhs
                        interp.stack.push(sum)
                case OpPrint:
                        val := interp.stack.pop()
                        fmt.Println(val)
                case OpLoadName:
                        pc++
                        // ...
                }
        }
}
```

Example - pygo/run

```
$> pygo
3
```

Victory!

Conclusions & Plans

Full code here: `github.com/sbinet/pygo`

Much more to implement and understand:

- functions (definition, call)
- frames, blocks
- closures, classes, ...
- REPL

Might migrate the production-grade code under `github.com/go-python`
Backport gained knowledge into the `go-interpreter` design.
Use it for Jupyter (`github.com/gopherds/gophernotes`)

See you on slack `#go-interpreter`?