

State machines in Go (#golang)

February 10, 2013

I've been rewriting critical server components that were originally written in [Python](#) to use [Go](#) instead. Unlike Python, which is interpreted and uses a [global lock because the interpreter is not thread-safe](#), Go has built-in support for concurrency, and is statically compiled. One of the first things I tackled was implementing a [state machine](#). The [Python version](#) was based on [this article by David Mertz](#). Mertz uses an object oriented approach, defining a class with both data and methods. His code, syntax aside, will be familiar to anyone who has worked with objects in C++, C#, and Java. Go, however, does not provide a mechanism for coupling methods to specific data structures. Instead, Go allows you to [associate methods with data structures](#), so that any method can be applied to any struct. It's a model which is closer to [what Alan Kay meant](#) when he defined the term object oriented in the first place. With that in mind, here's how I originally wrote the state machine class as a Go struct:

```
type Machine struct {
    Handlers map[string]func(interface{}) (string, interface{})
    StartState string
    EndStates map[string]bool
}
```

Just as in Mertz's definition, `Handlers` is a map whose keys are name strings, and whose values are functions which accept a "cargo" value, and return a next state name string, along with the updated cargo value. Go treats functions as first-class objects, so storing and passing them from state to state works exactly as it does in Python. The only change I made was in the end state list: whereas Mertz used a list of strings, I used a map, since there's no fast equivalent of detecting presence in a list (in Go, the only way to do it would be to iterate through the entire list of strings until or unless a match is found). Since the handler function signature is a little unwieldy, I created a [user-defined function type](#) for it:

```
type Handler func(interface{}) (string, interface{})

type Machine struct {
    Handlers map[string]Handler
    StartState string
    EndStates map[string]bool
}
```

All that leaves is the definition of the methods associated with the `Machine` struct. The first two provide a way to define which handler function is associated with which name string, and which name strings constitute end states:

```
func (machine *Machine) AddState(handlerName string, handlerFn Handler) {
    machine.Handlers[handlerName] = handlerFn
}


func (machine *Machine) AddEndState(endState string) {
    machine.EndStates[endState] = true
}
```

It's worth noting that since `EndStates` is a map (or a list, in Mertz's original), it's possible to have more than one state terminate processing. The final method is the one which executes the state machine, by applying the appropriate handler function to the cargo value, stopping when an end state has been reached. Since the set of functions are stored as first-class objects in a map, looking them up based on their names and invoking them is trivial:

```
func (machine *Machine) Execute(cargo interface{}) {
    if handler, present := machine.Handlers[machine.StartState]; present {
        for {
            nextState, nextCargo := handler(cargo)
            _, finished := machine.EndStates[nextState]
            if finished {
                break
            } else {
                handler, present = machine.Handlers[nextState]
                cargo = nextCargo
            }
        }
    }
}
```

The only possible fly in the ointment is Go's strong typing, since the type of the cargo value in the handler function signature needs to be specified. By using the generic `interface{}` as the type, however, all the handler functions need to do is invoke a [type assertion](#) on the incoming cargo value, and they can handle any data (the [test example](#) uses a float for the cargo, but it can be any data type, even user-defined structs). The complete state machine is [available as a Go package](#).

Archived from the original at <http://denis.papathanasiou.org/>

 Bitcoin Donate: [14TM4ADKJbaGEi8Qr8dh4KfPBQmjTshkZ2](https://www.blockchain.com/transaction/14TM4ADKJbaGEi8Qr8dh4KfPBQmjTshkZ2)