Flow-based programming

In Golang

25 April 2017

Thuc Le
Software Engineer

Definition

Flow-based Programing is a programing paradigm that defines applications as networks of **black box processes** which exchange data across **predefined connections** by **message passing**, where connections are **specified externally** to the processes.
- Paul Morrison

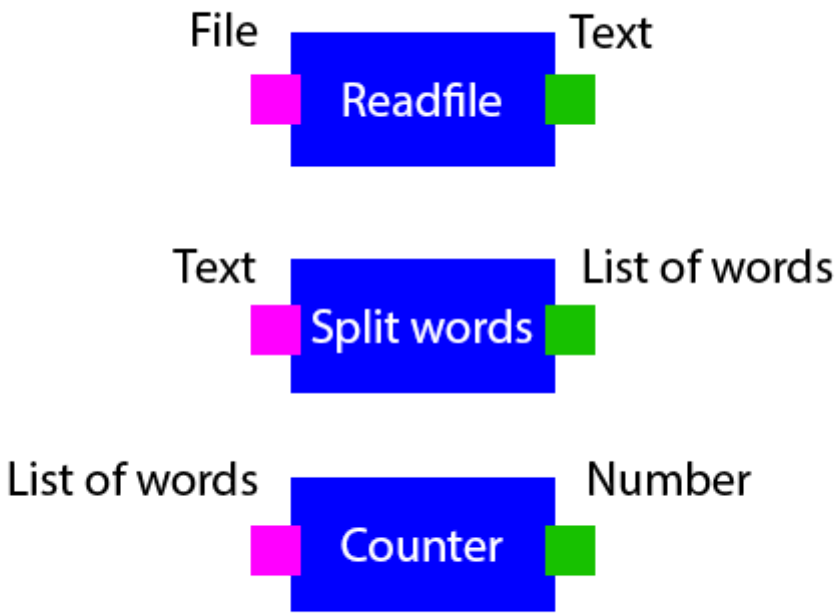
Core concepts

Process

- The process is black box
- The box doesn't care the outside world

Ports

- Way the processes communicate with outside world
- A process can have one or many inputs/outputs ports
- The "black box" process looks like data factory

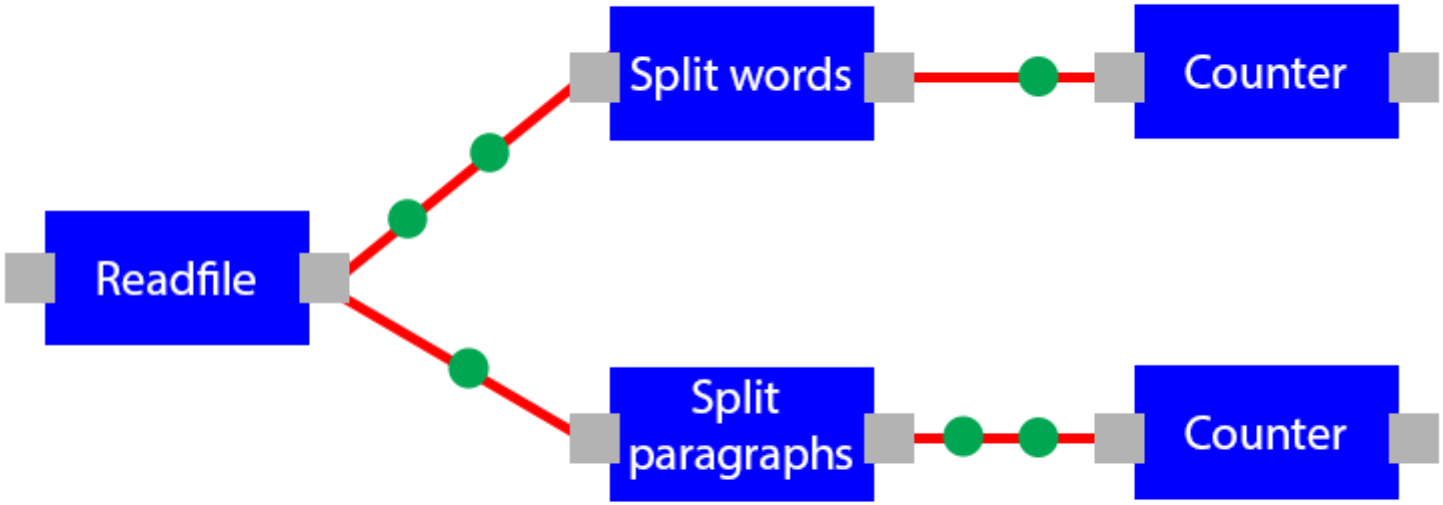


Connection

- The connection is defined externally to the processes
- The blackbox processes can be reuse in other application by re-conected without any changes.
- The connections work as streams of structured data chunks, called information packages (IPs)

Other notes:

- More than one process can execute the same piece of code
- Processes work asynchronously



Benefits:

- Improve the workflow visualization
- When develop the process, we focus on components development. It's easier for coding and testing.
- When develop the network, we focus on application's flow.
- FBP applications generally run in less elapsed time than conventional programs

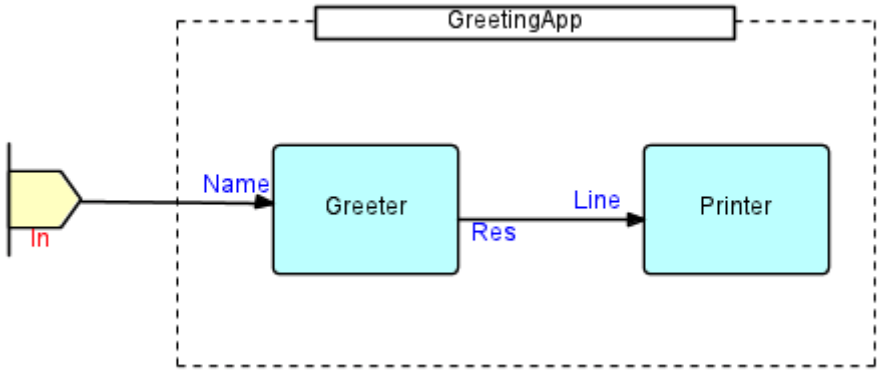
Drawback

- Complex the simple logic
- Take time of CPU to setup of the network of processes before running

Goflow

Goflow

- A library of golang to implement the Flow-based programming model.
- https://github.com/trustmaster/goflow
- Try with simple application:



Goflow - Define processes

```
9  // START GREETING
10
11  type Greeter struct {
12      flow.Component
13      Name <-chan string // input port
14      Res  chan<- string // output port
15
16      counter int
17  }
18
19  func (g *Greeter) OnName(name string) {
20      counter = counter + 1
21      greeting := fmt.Sprintf("Hello, %s!", name)
22      g.Res <- greeting //send the greeting to the output port/
23  }
24
25  // END GREETING
```

Goflow - Define processes

```
27  // START PRINTER
28
29  type Printer struct {
30      flow.Component
31      Line <-chan string // inport
32  }
33
34  func (p *Printer) OnLine(line string) {
35      fmt.Println(line)
36  }
37
38  // END PRINTER
```

Goflow - Networking

```
40  // START NETWORKING
41
42  type GreetingApp struct {
43      flow.Graph
44  }
45
46  func NewGreetingApp() *GreetingApp {
47      // Init application
48      n := new(GreetingApp)
49      n.InitGraphState()
50
51      // Add processes into application
52      n.Add(new(Greeter), "greeter")
53      n.Add(new(Printer), "printer")
54
55      // Create the connection
56      n.Connect("greeter", "Res111", "printer", "Line")
57
58      // Create the Application port
59      n.MapInPort("In", "greeter", "Name")
60      return n
61  }
62
63  // END NETWORKING
```

Goflow - Run

```
69  // START RUN
70
71  func main() {
72      // Create the application
73      net := NewGreetingApp()
74
75      // Open the application port `In`
76      in := make(chan string)
77      net.SetInPort("In", in)
78      defer close(in)
79
80      // Run the application and push the name
81      flow.RunNet(net)
82      in <- "Thuc Le"
83      in <- "Lazada"
84
85      // Wait until the app has done its job
86      <-net.Wait()
87  }
88
89  // END RUN
```

Goflow - Benchmark

- Run normally: 1.8-1.9 seconds
- Run goflow: 1.5-1.7 seconds

Goflow - Concurrency model

Goflow 3 different concurrency models at process level
- Asynchronous
- Synchronous
- Pool of workers

```
greeter := new(Greeter)

// Sync mode
greeter.Component.Mode = flow.ComponentModeAsync

// Sync mode
greeter.Component.Mode = flow.ComponentModeSync

// Pool of workers mode
greeter.Component.Mode = flow.ComponentModePool
greeter.Component.PoolSize = 8 // 8 workers in the crew
```

Goflow - State lock

- Support built-in lock in process

```
type StateExample struct {
  flow.Component

  In1 <-chan string

  // lock
  StateLock *sync.Mutex

  // state
  counter int
  buffer  [32]string
}

func (p *StateExample) OnIn2(item String) {
  // We can safely modify fields here
  p.buffer = append(item, s)
  p.counter++
}
```

Goflow - Types of port

- Goflow supports 3 types of ports: Input port, output port bi-direction port

```
type PortExample struct {
  flow.Component
  Foo <-chan int  // receive-only inport
  Bar chan<- int  // send-only outport
  Boo chan string // can be inport or outport
}
```

- Support method raise when a connection is closed

```
func (p *Printer) OnLine(line string) {
  fmt.Println(line)
}

func (c *ComponentType) OnLineClose() {
  // Clean something
}
```

Thank you

Thuc Le
Software Engineer
ledongthuc@gmail.com (mailto:ledongthuc@gmail.com)