A tour of YAML parsers for Go

Michel Casabianca
casa@sweetohm.net

Here is a list of YAML parsers for Go programming language. For each one, I will provide a short description, an evaluation and an example source code.

Gopher

You can download an archive with example source code here.

Note that names of the parsers were customized by myslef as they are almost all named **goyaml** :o)

In the code example, we will parse following YAML file:

```
foo: 1
bar:
  - one
  - two
```

In each source, we will extract and print the first element of the list in *foo* key of the map at the root of the document.

Goyaml

- Homepage: http://gopkg.in/yaml.v2.
- Documentation: http://godoc.org/gopkg.in/yaml.v2.

This parser is very handy to manage configuration files. You declare a struct that maps the structure of the YAML configuration file and you pass a reference to this struct to the parser while loading configuration file. This parser will populate struct with data in the file.

This parser is probably the best way to manage YAML configuration files.

Example

```
package main

import (
```

```
    "fmt"
    "gopkg.in/yaml.v2"
    "io/ioutil"
    "os"
)

type Config struct {
    Foo string
    Bar []string
}

func main() {
    filename := os.Args[1]
    var config Config
    source, err := ioutil.ReadFile(filename)
    if err != nil {
        panic(err)
    }
    err = yaml.Unmarshal(source, &config)
    if err != nil {
        panic(err)
    }
    fmt.Printf("Value: %#v\n", config.Bar[0])
}
```

To run this sample code, type following command:

```
go run src/goyaml.go src/test.yml
```

Gypsy

- Homepage: https://github.com/kylelemons/go-gypsy.
- Documentation: https://godoc.org/github.com/kylelemons/go-gypsy/yaml.

This is a low level parser, which means that a file will result in a nodes tree that you will have to explore by hand (as in the example that follows):

Example

```
package main

import (
    "fmt"
    "github.com/kylelemons/go-gypsy/yaml"
    "os"
)

func nodeToMap(node yaml.Node) (yaml.Map) {
    m, ok := node.(yaml.Map)
    if !ok {
        panic(fmt.Sprintf("%v is not of type map", node))
```

```
    }
    return m
}

func nodeToList(node yaml.Node) (yaml.List) {
    m, ok := node.(yaml.List)
    if !ok {
        panic(fmt.Sprintf("%v is not of type list", node))
    }
    return m
}

func main() {
    filename := os.Args[1]
    file, err := yaml.ReadFile(filename)
    if err != nil {
        panic(err)
    }
    value := nodeToList(nodeToMap(file.Root)["bar"])[0]
    fmt.Printf("Value: %#v\n", value)
}
```

To run this sample code, execute following command:

```
go run src/gypsy.go src/test.yml
```

Bug

Gypsy parser won't parse following YAML file (because of the indentation of the list which is at the same level than the enclosing map):

```
foo: 1
bar:
- one
- two
```

Beego

- Homepage: https://github.com/beego/goyaml2.
- Documentation: http://godoc.org/github.com/beego/goyaml2.

Beego is a build tool, but it includes a low level YAML parser. Mappings are very natural with this parser:

- A YAML *map* maps to Go *map[string]interface{}*.
- A YAML *list* maps to Go *[]interface{}*.
- and so on...

Example

```
package main

import (
    "fmt"
    "github.com/beego/goyaml2"
    "os"
)

func nodeToMap(node interface{}) map[string]interface{} {
    m, ok := node.(map[string]interface{})
    if !ok {
        panic(fmt.Sprintf("%v is not of type map", node))
    }
    return m
}

func nodeToList(node interface{}) []interface{} {
    m, ok := node.([]interface{})
    if !ok {
        panic(fmt.Sprintf("%v is not of type list", node))
    }
    return m
}

func main() {
    filename := os.Args[1]
    file, err := os.Open(filename)
    if err != nil {
        panic(err)
    }
    object, err := goyaml2.Read(file)
    if err != nil {
        panic(err)
    }
    value := nodeToList(nodeToMap(object)["bar"])[0]
    fmt.Printf("Value: %#v\n", value)
}
```

Golly

- Homepage: http://github.com/tav/golly.
- Documentation: http://godoc.org/github.com/tav/golly/yaml.

This parser is part of a general Go library. This may only parse dictionnaries and its API is a bit tedious.

Example

```
package main

import (
```

```
    "fmt"
    "github.com/tav/golly/yaml"
    "os"
)

func main() {
    filename := os.Args[1]
    data, err := yaml.ParseFile(filename)
    if err != nil {
        panic(err)
    }
    list, ok := data.GetList("bar")
    if !ok {
        panic("Not OK!")
    }
    value := list[0].String
    fmt.Printf("Value: %#v\n", value)
}
```

Simple YAML

- Homepage: https://github.com/smallfish/simpleyaml.
- Documentation: http://godoc.org/github.com/smallfish/simpleyaml.

This is a layer around GoYAML to simplify YAML file parsing. You don't have to describe document structure, you can directly access content exploring the tree. Nevertheless, what you save on one side (not writing the document structure), you loose on the other (writing code to explore the tree and managing errors)...

Example

```
package main

import (
        "fmt"
        "github.com/smallfish/simpleyaml"
        "io/ioutil"
        "os"
)

func main() {
        filename := os.Args[1]
        source, err := ioutil.ReadFile(filename)
        if err != nil {
                panic(err)
        }
        yaml, err := simpleyaml.NewYaml(source)
        if err != nil {
                panic(err)
        }
        bar, err := yaml.Get("bar").GetIndex(0).String()
        if err != nil {
                panic(err)
```

```
        }
        fmt.Printf("Value: %#v\n", bar)
}
```

Zombiezen

- Homepage: https://bitbucket.org/zombiezen/yaml.
- Documentation: No known documentation.

The body of the `Parser` struct in a TODO comment... Thus seems incomplete and unusable.