Narrative

Rob Rohan

2022-04-08

Abstract

"Narrative is a technique to comment code to produce both the code itself, and documentation / a manual / a tutorial about the code. It is a similar idea to literate programming, but focused more on the code than on producing an academic paper."

Contents

1 Introduction

Narrative is a technique to comment code to produce both the code itself, and at the same time create documentation (a manual, a tutorial, etc) about the code.

It is a similar idea to, and was inspired by, literate programming. Narrative is more focused on guiding someone through the code than on producing an academic paper.

View example output from this codebase (work in progress)

2 Includes

Here we are importing several packages. Since this application mostly just combines files and concatenates strings, almost all of these imports are to support those activities.

The *ardanlabs/conf* include is a library to help make using command line parameters a bit easier. More can be seen on their website …

(Web of Stories - Life Stories of Remarkable People 2016)

```
package main

import (
   "bufio"
   "fmt"
   "io"
   "log"
   "os"
   "path/filepath"
   "strings"

   "github.com/ardanlabs/conf"
)
```

This *build* variable will be overwritten by our build script. This value will be the git hash of the current, build time commit.

```
var build = "develop"
```

3 Config Struct

This structure is used to hold the command line parameters passed when the application was started. Note that only *input* is required, and *input* needs to be a line sparated file that has a list of files to concatenate together.

```
type Config struct {
   Start  string `conf:"short:s,default:/*"`
   End    string `conf:"short:e,default:*/"`
   Input  string `conf:"short:i,required"`
   Output string `conf:"short:o,default:final.md"`
}
```

4 Parse the NARRATIVE File

```
func parseHeader(cfg Config, log *log.Logger) {
   // the narrative config file with the list of files we'll parse
   narrativeFile, err := os.Open(cfg.Input)
   if err != nil {
       log.Fatal(err)
   }
```

```
    defer narrativeFile.Close()

    // output file
    fout, err := os.OpenFile(cfg.Output, os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        log.Fatal(err)
    }
    defer fout.Close()

    dir := filepath.Dir(narrativeFile.Name())
    rd := bufio.NewReader(narrativeFile)
    for {
        line, err := rd.ReadString('\n')
        if err != nil {
            if err == io.EOF {
                break
            }
            log.Fatal(err)
            return
        }

        line = strings.Trim(line, "\n ")
        if line == "" || line[0] == '#' {
            continue
        }
        inputFile := fmt.Sprintf("%s%c%s", dir, filepath.Separator, line)
        log.Println(inputFile)
        parse(cfg, log, inputFile, fout)
    }
}
```

5 Parse a Markdown File

```
func parse(cfg Config, log *log.Logger, filePath string, fout io.Writer) {
    code_mode := false
    file, err := os.Open(filePath)
    if err != nil {
        log.Fatal(err)
        return
    }
    defer file.Close()

    extension := strings.ToLower(filepath.Ext(filePath))

    scanner := bufio.NewScanner(file)
    line := ""
    for scanner.Scan() {
        line = scanner.Text()

        if extension == ".md" || extension == ".markdown" {
            _, err := fmt.Fprintf(fout, "%s\n", line)
            if err != nil {
                log.Fatal(err)
            }
        } else {
            // line = strings.Trim(line, "\n ")
            if line == cfg.Start {
                code_mode = true
                continue
            }
            if line == cfg.End {
                code_mode = false
                continue
            }

            if code_mode {
                _, err := fmt.Fprintf(fout, "%s\n", line)
                if err != nil {
                    log.Fatal(err)
                }
            } else {
                _, err := fmt.Fprintf(fout, "    %s\n", line)
                if err != nil {
                    log.Fatal(err)
                }
            }
        }
    }
    if err := scanner.Err(); err != nil {
        log.Fatal(err)
    }

}
```

6 Run Wrapper

```
func run(log *log.Logger) error {
    cfg := Config{}
    if err := conf.Parse(os.Args[1:], "NT", &cfg); err != nil {
        if err == conf.ErrHelpWanted {
            usage, err := conf.Usage("NT", &cfg)
            if err != nil {
                return err
            }
            fmt.Println(usage)
            return nil
        }
        return err
    }

    parseHeader(cfg, log)

    return nil
}
```

7 Program Main Entry

```
func main() {
    log := log.New(os.Stdout, "NT: ", log.LstdFlags|log.Lmicroseconds|log.Lshortfile)

    if err := run(log); err != nil {
        log.Println("error: ", err)
        os.Exit(1)
    }
}
```

Web of Stories - Life Stories of Remarkable People. 2016. 'Donald Knuth - Literate Programming (66/97).'