

Directory Synchronization in Go

Probing Presentation

Zhibin Cai

Outline

1. Introduction
 2. Describing the probes
 3. Demos
 4. Conclusions
-

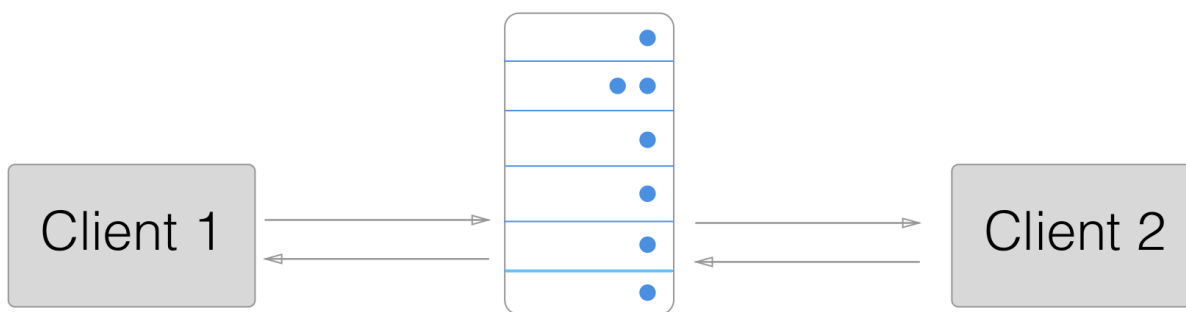
Introduction

Project Goal Review

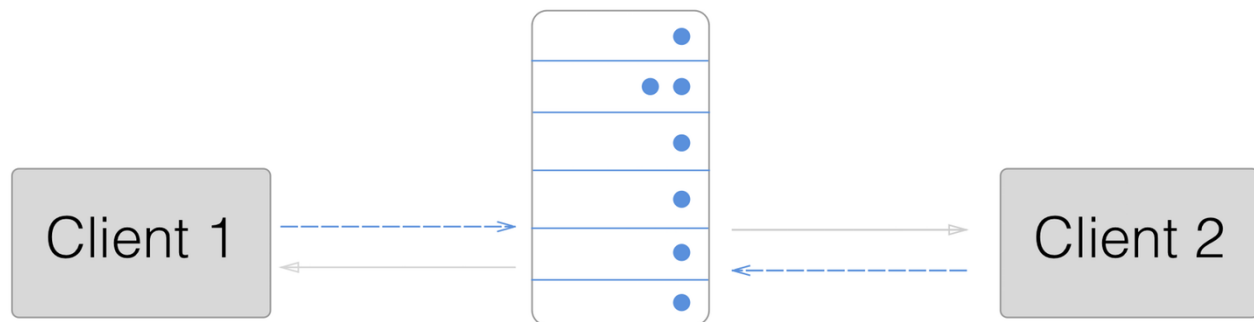
Generally speaking, I am creating another Dropbox that sync files across the network.

Definition

The goal is to create a software solution that has multiple clients and a centralized server, so that different clients can sync their files automatically, using the centralized server as a bridge.



Project structure



Problems that I have solved (sending files from clients to server)

Probes that I have done

- ☐ Project configuration loading (settings are written down in a config file)
- ☐ Directory Scanning
 - ☐ File watcher
 - ☐ A DB file describing the properties of each file in the synching directory
- ☐ Sending files over http
 - ☐ A client sending files
 - ☐ A server accepting requests and receiving files

Probes Description

Project configuration loading

The settings of this application are written in a config file (`config.toml`):

```
1 [development]
2 ServerAddr = "http://localhost"
3 Port = 8182
4 UploadAddrPattern = "/upload"
5 ServerFilesLocation = "./files/"
6 FileDbJsonPath = "./filedb.json"
7 SyncRoot = "$GOPATH/src/github.com/tonychol/sink/testDir"
8
9 [production]
10 ServerAddr = "http://localhost"
11 Port = 80
12 ...
```

```
1 import (
2     "log"
3     "errors"
4     "sync"
5     "github.com/spf13/viper"
6 )
7
8 // Config : A struct that holds a bunch of configurations
9 type Config struct {
10     DevServer          string
11     DevPort             int
12     DevUploadURLPattern string
13     ServerFilesLocation string
14     FileDbJSONPath      string
15     SyncRoot            string
16 }
17
18 var instance *Config
19 var once sync.Once
20
21 // GetInstance : Using singleton to get the global config instance
22 func GetInstance() *Config {
23     once.Do(func() {
24         validConfig, err := loadConfig()
25         if err != nil {
26             log.Fatal("Config not initialized")
27             panic(err)
28         } else {
29             instance = &validConfig
30         }
31     })
32     return instance
33 }
```

Smooth transition when deploying the product into **production**.

Directory Scanning

★ One of the most important parts.

The basic Idea is to create a file watcher for each directory.

1. File Watcher

▼ testDir	Today, 11:21 AM	--	Folder
▶ dir1 1	Today, 11:21 AM	--	Folder
▼ dir2 2	Apr 15, 2017, 3:20 PM	--	Folder
▶ dir3 3	Yesterday, 5:31 PM	--	Folder
▶ helloworld 4	Today, 5:47 AM	--	Folder
two.txt 5	Yesterday, 5:31 PM	39 bytes	Plain Text

File Watcher Example

▼ testDir	Today, 11:21 AM	--	Folder
▼ dir1	Today, 11:21 AM	--	Folder
▶ subdir 1	Yesterday, 1:09 PM	--	Folder
▶ subdir2 2	Apr 14, 2017, 2:15 PM	--	Folder
three.txt 3	Apr 15, 2017, 3:52 PM	Zero bytes	Plain Text
▼ dir2	Apr 15, 2017, 3:20 PM	--	Folder
▶ dir3	Yesterday, 5:31 PM	--	Folder
▶ helloworld	Today, 5:47 AM	--	Folder
two.txt	Yesterday, 5:31 PM	39 bytes	Plain Text

File Watcher in sub directory

We have one watcher in each directory(recursively).


[howeyc / fsnotify](#)

Watch ▾

89

★ Star

1,411

🔗 Fork

198

<> Code

🔔 Issues 18

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📶 Pulse

📊 Graphs

File system notification for Go

Cross platform file system notification library in Golang

An event is fired under these circumstances:

- `ev.IsCreate()` : creation of the file
- `ev.IsDelete()` : deletion of the file
- `ev.IsModify()` : modification of the file

- `ev.IsRename()` : renaming of the file
- `ev.IsAttrib()` : a change in the file's metadata

Code snippets for file watcher

```
1  import (
2      "log"
3      "os"
4      "github.com/howeyc/fsnotify"
5      "github.com/tonychol/sink/fs"
6      "github.com/tonychol/sink/scanner"
7      "github.com/tonychol/sink/util"
8  )
9  // watchDir launches a goroutine to watch all the directories
10 // and fires the specific file event
11 func watchDir(done chan bool, dirs ...string) {
12     watcher, err := fsnotify.NewWatcher()
13     util.HandleErr(err)
14     defer watcher.Close()
15     go func() {
16         for {
17             select {
18             case ev := <-watcher.Event:
19                 log.Println("event:", ev)
20                 eventFile := ev.Name
21                 if ev.IsCreate() {
22                     // Watch new file if it is directory
23                     // ...
24                 }
25                 if ev.IsDelete() {
26                     // Stop watching this file because it is deleted
27                     watcher.RemoveWatch(eventFile)
28                 }
29                 if ev.IsModify() {
30                 }
31                 if ev.IsRename() {
32                 }
33                 if ev.IsAttrib() {
```

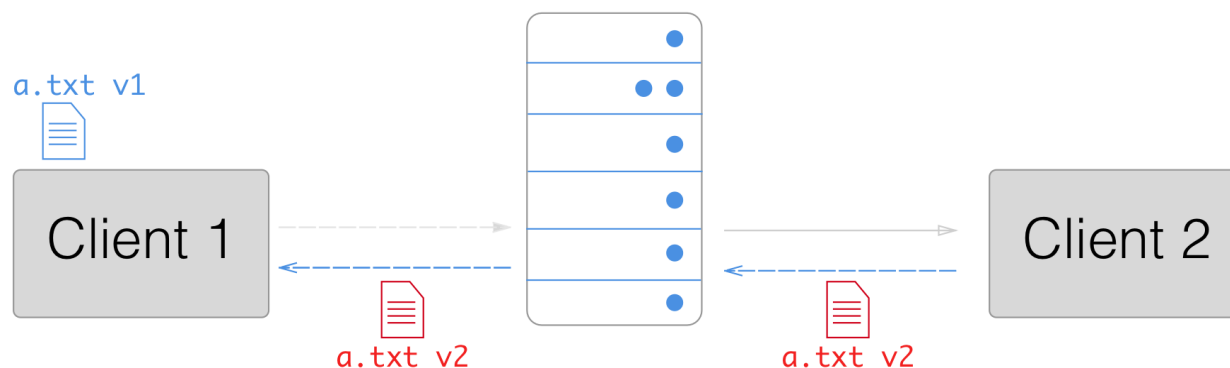
```

34     }
35     case err := <-watcher.Error:
36         log.Println("error", err)
37         done <- true
38     }
39 }
40 }()
41 }

```

2. DB file describing the properties of each file

The goal of this db file is to store the information about each file that might be used to compare.



Version comparison

```

λ ~/Dropbox/ ls -a | grep dropbox
.dropbox
.dropbox.cache
λ ~/Dropbox/

```

The caches and db file in Dropbox's folder

```

1  {
2    "/Users/tonychol/go/src/github.com/tonychol/sink/testDir": {
3      "FileType": "d",
4      "Mode": 2147484141,
5      "Checksum": "",
6      "LastModify": "2017-04-20T05:46:59-04:00"
7    },
8    "/Users/tonychol/go/src/github.com/tonychol/sink/testDir/.DS_Store": {
9      "FileType": "f",
10     "Mode": 420,
11     "Checksum": "0f07ef442af49e974a322b86113f619b",
12     "LastModify": "2017-04-15T17:41:12-04:00"
13   },
14   "/Users/tonychol/go/src/github.com/tonychol/sink/testDir/anotherDir2": {
15     "FileType": "d",
16     "Mode": 2147484141,
17     "Checksum": "",
18     "LastModify": "2017-04-19T13:09:22-04:00"
19   },
20   "/Users/tonychol/go/src/github.com/tonychol/sink/testDir/anotherDir2/hello": {
21     "FileType": "d",
22     "Mode": 2147484141,
23     "Checksum": "",
24     "LastModify": "2017-04-19T13:09:22-04:00"
25   },

```

Parts of the persisted db file (JSON format)

```

1  // FileDBElement is the element of the db that
2  // describes the information of the synching directory
3  type FileDBElement struct {
4      FileType    string
5      Mode        os.FileMode
6      CheckSum    string
7      LastModify  time.Time
8  }
9
10 // FileDB is a map represents the information of each file among the whole d
11 // irectory
12 // key : each file's path string
13 // val : The FileDBElement struct that holds information
14
15 type FileDB map[string]FileDBElement
16
17 // Singleton
18
19 var instance *FileDB
20 var once sync.Once
21
22 // GetFileDBInstance : Using singleton to get the global filedb instance
23
24 func GetFileDBInstance() *FileDB {
25     once.Do(func() {

```

```
22     validConfig := restoreDBFromJSONFile()
23     instance = &validConfig
24 })
25 return instance
26 }
```

```
1 // restoreDBFromJSONFile loads the db instance from the json file
2 // if this file does not exists, then return an empty FileDB instance.
3 // Note that this function will only be called by `GetFileDBInstance()`
4 func restoreDBFromJSONFile() FileDB {
5     absPath, err := filepath.Abs(config.GetInstance().FileDbJSONPath)
6     util.PanicIf(err)
7     dat, err := ioutil.ReadFile(absPath)
8     if err != nil {
9         return make(FileDB)
10    }
11    var dbInstance FileDB // map: [string] -> FileDBElement
12    err = json.Unmarshal(dat, &dbInstance)
13    util.PanicIf(err)
14    return dbInstance
15 }
```

The end of directory scanning

Next part is to send the files from clients to server.   

File Transfer

Two parts:

1.  A client sending files
 2.  A server accepting requests and receiving files
-

Server side

A RESTful server is written in Go and launched in an Ubuntu virtual machine.

```
1 import "http"
2 // Server's main function
3 func main() {
4     http.HandleFunc("/upload", upload)
5     log.Println("Server has been set up at :8181")
6     err := http.ListenAndServe(":8181", nil) // set listen port
7     util.HardHandleErr(err)
8 }
```

```
1 // upload logic: parses a request body as multipart/form-data.
2 func upload(w http.ResponseWriter, r *http.Request) {
3     fmt.Println("method:", r.Method)
4     if r.Method == "POST" {
5         relativePath := r.FormValue("relativePath")
6         filename := r.FormValue("filename")
7         r.ParseMultipartForm(32 << 20)
8         file, handler, err := r.FormFile("uploadfile")
9         if err != nil {
10             fmt.Println(err)
11             return
12         }
13         defer file.Close()
14         fmt.Fprintf(w, "%v", handler.Header)
15         // Try to create the directory to hold the target incoming file if not e
16         xist
17         err = createDirIfNotExist(relativePath)
18         util.HardHandleErr(err)
19         targetFilePath := baseDir + relativePath + string(filepath.Separator) +
20 filename
21         f, err := os.OpenFile(targetFilePath, os.O_WRONLY|os.O_CREATE, 0666)
22         if err != nil {
23             fmt.Println(err)
24             return
25         }
26     }
```

```
24     defer f.Close()
25     io.Copy(f, file)
26 }
27 }
```

Client side

The goal is to use http `POST` method to post form data to the server with the file attached.

```
1 // postFile accepts a filename which is a relative path
2 // and its targetUrl of the server, posts the file to the server
3 func postFile(filename string, targetURL string) error {
4     bodyBuf := &bytes.Buffer{}
5     bodyWriter := multipart.NewWriter(bodyBuf)
6     relativePath, err := fs.GetRelativeDirFromRoot(filename)
7     util.HardHandleErr(err)
8     fileFullName := fs.GetFileNameFromFilePath(filename)
9     // this step is very important
10    bodyWriter.WriteField("relativePath", relativePath)
11    bodyWriter.WriteField("filename", fileFullName)
12    fileWriter, err := bodyWriter.CreateFormFile("uploadfile", filename)
13    if err != nil {
14        fmt.Println("error writing to buffer")
15        return err
16    }
17    // open file handle
18    fh, err := os.Open(filename)
19    util.HardHandleErr(err)
20    //iocopy
21    _, err = io.Copy(fileWriter, fh)
22    util.HardHandleErr(err)
23    contentType := bodyWriter.FormDataContentType()
24    bodyWriter.Close()
25    resp, err := http.Post(targetURL, contentType, bodyBuf)
26    if err != nil {
27        return err
28    }
```

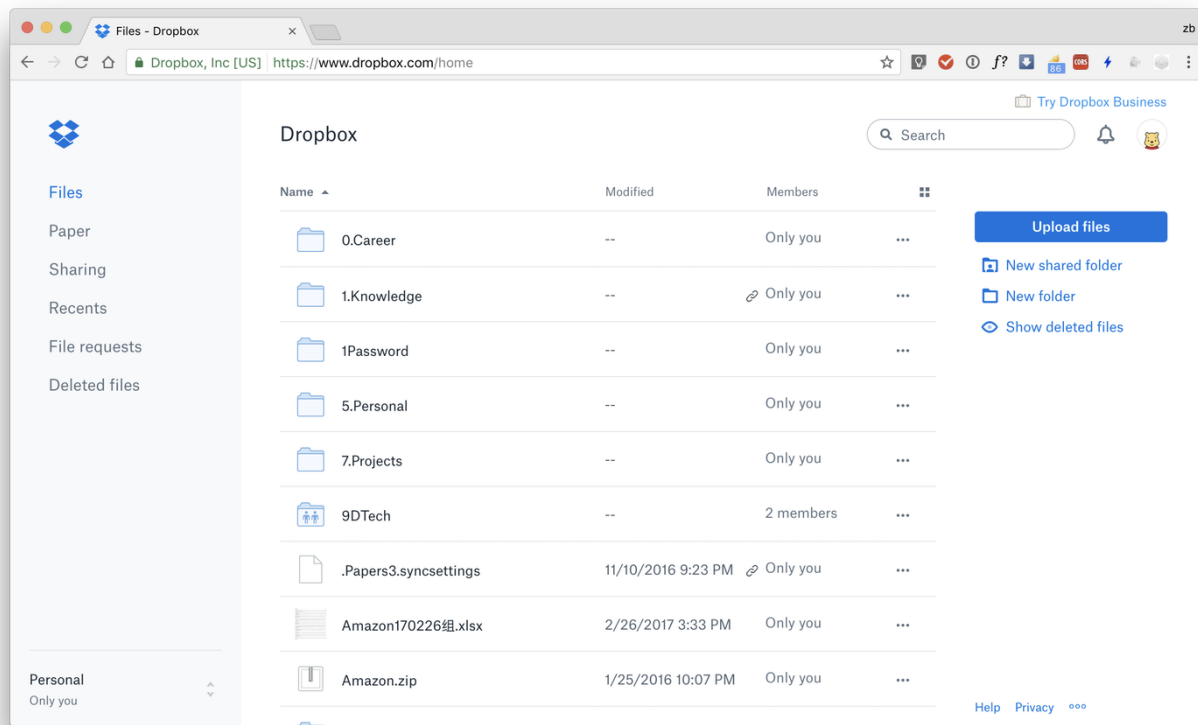
```
29     defer resp.Body.Close()
30     respBody, err := ioutil.ReadAll(resp.Body)
31     if err != nil {
32         return err
33     }
34     fmt.Println(resp.Status)
35     fmt.Println(string(respBody))
36     return nil
37 }
```

Demo

Future plan

Milestones to go:

- ☐ Send files from server to multiple clients
 - ☐ Using Socket or [WebSocket](#)
 - ☐ Whenever a new file coming to server, spread the file out to every connected client
- ☐ Syncing policy
 - ☐ How to handle conflicts? [Dropbox's policy](#)
- ☐ Web version of this product
 - ☐ Server is already RESTful, which is great.
 - ☐ Multiple users and authentication support



Conclusions

What did I learned?

1. Go is an interesting programming language.
 - a. In the first presentation, I got too attached to the concurrency...
 - b. There are so many things in Go that surprises me
 - i. **Error handling (see the code example)**

```
1 if err != nil {  
2     fmt.Println("error writing to buffer")  
3     return err  
4 }  
5 // open file handle  
6 fh, err := os.Open(filename)  
7 if err != nil {  
8     fmt.Println("error opening file")  
9     return err
```

```
10 }
11 //iocopy
12 _, err = io.Copy(fileWriter, fh)
13 if err != nil {
14     return err
15 }
16 contentType := bodyWriter.FormDataContentType()
17 bodyWriter.Close()
18 resp, err := http.Post(targetURL, contentType, bodyBuf)
19 if err != nil {
20     return err
21 }
```

Single point of failure

- a. The client side of the application could crash
- b. The client system may have h/w problems
- c. The client's network card could fail
- d. Network contention could cause timeouts
- e. There may be network address conflicts
- f. Network elements such as routers could fail
- g. Transmission errors may lose messages
- h. The client and server versions may be incompatible
- i. The server's network card could fail
- j. The server system may have h/w problems
- k. The server s/w may crash
- l. The server's database may become corrupted

What did I learned?

2. The tool chain of Go is very convenient
 - a. Testing is easy (go checkout the editor)
 - b. Source code of Go standard library is also readable...

What did I learned?

3. I would choose to write Go program more in my future projects
 - You write Go program smoothly with great development speed, just like Python,
 - but with type support, great compilation, powerful concurrency model
 - and also great performance.

4. Highly recommend it! 👍

Thank you!

Reference:

1. Network Programming with Go (golang) by [Jan Newmarch](#)
2. GitHub Project: [howeyc/fsnotify](#) for file system notification
3. GitHub Project: [spf13/viper](#) for `toml` style configuration