测试报告

Naive gDocs

Distributed File System

分布式文件系统的测试分为以下几个部分

锁

对 lock.go 中自行实现的自旋锁的测试：
client/testlock.go

通过产生了三个不同的协程分别去连接 zookeeper，并尝试 lock 相同的一个节点，每人锁住之后睡眠 3 秒，再放锁。

一个协程的示例代码：

```
go func() {
    i:=1
    fmt.Println(strconv.Itoa(i)+"test")
    connection := connect()
    fmt.Println(strconv.Itoa(i)+" connect success")
    lock(connection,"/test-lock")
    fmt.Println(strconv.Itoa(i)+" lock success")
    time.Sleep(time.Second * 3)
    unlock(connection,"/test-lock")
    fmt.Println(strconv.Itoa(i)+" unlock success")
    close(connection)
    wg.Done()
}()
```

通过检查命令行的输出，发现多次测试中每次测试都不会有 lock success 交叉出现。
即每一个协程锁住后别人都不能再锁，成功地达到了防止不同协程进入同一个 critical section 的效果。

单协程读写

对文件系统的读写，通过使用 gfs_api 中调用文件系统的接口，成功达到读写效果。

读写测试的示例代码：

```
    fmt.Println("Create test begin!")
    create("test-02")
    write("test-02", 13, "test1test2test3test4")
    result := read("test-02", 16, 8)
    fmt.Println(result)
    //deletee("test-02")
    fmt.Println("Create test end!")
```

检查命令行的输出，读出的与写入的一致，成功的达到了单协程读写测试的目的。

多协程并发

我们创建十个文件，并让一百个协程以十个为一组分别写同一个文件，然后读，我们发现结果和预期一致

```
wg := sync.WaitGroup{}
wg.Add(10)
fmt.Println("multi create test begin")
prefix := "test-"
for i := 0; i < 10; i++ {
    filename := prefix + strconv.Itoa(i)
    go func() {
        create(filename)
        wg.Done()
    }()
}
wg.Wait()

fmt.Println("multi create test end")
}
```

```
wg := sync.WaitGroup{}
wg.Add(100)
fmt.Println("multi write test begin")
prefix := "test-"
for i := 0; i < 10; i++ {
    filename := prefix + strconv.Itoa(i)
    for j := 0; j < 10; j++ {
        temp := j
        go func() {
            content := "nanaminanami"
                        write(filename, uint64(temp*15), content)
            wg.Done()
        }()
    }
}
wg.Wait()

fmt.Println("multi write test end")


wg := sync.WaitGroup{}
wg.Add(10)
fmt.Println("multi read test begin")
prefix := "test-"
for i := 0; i < 10; i++ {
    filename := prefix + strconv.Itoa(i)
    go func() {
        result := read(filename, 0, 200)
        fmt.Println("From ", filename, " we read ", result)
        wg.Done()
    }()
}
wg.Wait()

fmt.Println("multi read test end")
```

性能测试

我们把write改为对每个文件写十次

```
for i := 0; i < 10; i++ {
    filename := prefix + strconv.Itoa(i)
    for j := 0; j < 10; j++ {
        content := "nanaminanami"
        write(filename, uint64(j*15), content)
    }
}
```

```
date
for i in `seq 1 1`
do
{
    go run ./gfs_api.go ./json_struct.go ./lock.go ./testMultiCreate.go
}&
done
wait
date
for i in `seq 1 1`
do
{
    go run ./gfs_api.go ./json_struct.go ./lock.go ./testMultiWrite.go
}&
done
wait
date
for i in `seq 1 1`
do
{
    go run ./gfs_api.go ./json_struct.go ./lock.go ./testMultiRead.go
}&
done
wait
date
```

四次时间的输出为：
2021年 7月17日 星期六 21时41分44秒 CST
2021年 7月17日 星期六 21时41分44秒 CST
2021年 7月17日 星期六 21时42分00秒 CST
2021年 7月17日 星期六 21时42分02秒 CST
可以看到10次create、100次write、10次read的时间分别为<1s，16s，2s

Frontend

通过对前端每一个组件的接口进行单元测试来完成前端测试

登录：

1. 输入用户名“a"，登录后，localStorage中包含Item User，对应值为”a". 在所有其他页面Json.parse User都能成功获取正确的用户名。

文件列表：

1. connect后，正确显示文件列表，内容与后端发回数据一致。
2. create file在后端消息发回前正常阻塞，根据返回消息类型报错返回，或成功创建文件。
3. create file后，新文件正常转发至所有前端进行更新。
4. delete file在后端消息发回前正常阻塞，根据返回消息类型报错返回，或成功将文件移动到回收站。
5. delete file后，文件被删除并正确转发至所有前端更新。
6. 点击文件系统Log后显示所有Log，与后端发来消息一致。
7. 点击回收站能正常切换至回收站页面。
8. 点击进入文件并进入后，能正常切换至文件的Sheet页面。

回收站：

1. 永久删除后，正确发送消息到后端，收到ret后所有前端更新回收站。
2. 恢复后，正确发送消息到后端，收到ret后所有前端更新回收站。
3. 点击文件列表能正确切换至文件列表页面。

Sheet页面：

1. load 表格，前端数据显示与后端发来数据一致，cell的值正常显示，并且拿锁信息正确显示。
2. 双击cell编辑并更新，文件系统正确写入。
3. 双击进入edit后，其他前端收到消息显示someone is editing信息，并更新前端锁，阻止其他人获取cell写权限。
4. 其他人试图进入edit会收到报错信息，并且无法更新信息到文件系统里（无锁无法写入）。
5. update后，放送信息给后端并且正常放掉前端锁。
6. 空表rollback显示，无法rollback报错信息。
7. 正常rollback已写内容。
8. 用户已写内容被其他用户编辑时，正确返回rollback内容正在被编辑的报错信息。
9. 用户已写内容被其他用户编辑后，正确返回rollback内容已被编辑过的报错信息。

Backend

通过实现测试client与webserver进行交互来实现对后端的测试

在后端的测试中，我们对后端的每个接口进行测试，通过查看后端是否能按照预期输出结果并返回正确的值来测试借口的功能

Create

测试正常创建文件file1与创建重复的文件file1并查看返回结果

测试结果输出如下：

```
// 创建成功，data对应当前文件列表
2021/07/17 13:45:38 recv: {"type":"CREATESUCCESS","data":["file1"]}
// 创建失败，CEF为创建重复的文件
2021/07/17 13:45:39 recv: {"type":"CEF","data":null}
```

符合预期

Open

测试打开文件file1，读取文件中的信息与当前文件的拿锁信息

测试结果输出如下

```
// 读取成功，data为当前文件每个单元格中的信息
2021/07/17 14:01:00 recv: {"type":"FileDataSlice","data":["","","","","","","","","","","","","","","","","","
......
","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","","
// 读取成功，data为当前文件被其他前端获取锁的信息
2021/07/17 14:01:01 recv: {"type":"FILELOCKINFO","data":[]}
```

符合预期

Edit

测试edit接口，当选中某个单元格时尝试获取对应单元格的锁，以下输出为后端输出，可见其成功获取了对应单元格的锁

```
[Edit] try to edit file:  file1
[EDIT] try to get lock: /file1-1-1
nnnnnnnnnnnotexist
[LOCKMAP] map[/file1-1-1:xzl]
[EDIT] get the lock for: /file1-1-1
```

当后端接收到editing信息并确定其成功获取锁后，会将信息转发给所有参与本文件编辑的前端，因此前端输出如下：

```
2021/07/17 14:12:24 recv: {"type":"editing","row":1,"column":1,"newValue":"data1","fileName":"file1","userName":'
```

结果符合预期

此时开启另外一个终端发送同样的editing请求，获取输出如下:

```
2021/07/17 14:16:10 recv: {"type":"FAILOCK","row":1,"column":1,"successUsername":"xzl","rejectUsername":"zyt"}
```

即获取锁失败，此时无法进行编辑，结果符合预期

Update

测试update接口，update会更新对应单元格的值并将edit阶段获取的锁释放，在成功进行update之后会将信息发送给所有参与文件编辑的前端，便于其及时更改对应单元格的信息，前端输出如下：

```
2021/07/17 14:28:24 recv: {"type":"update","row":1,"column":1,"newValue":"data1","fileName":"file1","userName":"›
```

可见已经成功将更新信息发送给前端，结果符合预期

查看后端webserver输出：

```
[UPDATE] unlock the lock: /file1-1-1
[UPDATE] write back to a connection
[UPDATE] success write back message {"type":"update","row":1,"column":1,"newValue":"data1","fileName":"file1","us
```

可见已经成功更新文件并放锁，结果符合预期

Move to Trash

测试移动到回收站接口，该操作会将文件移动到回收站

测试将前面创建的file1文件移动至回收站

测试前端接受返回值如下：

```
021/07/17 14:38:32 recv: {"type":"MTTSUCCESS","data":["file"]}
```

可见成功将文件移动至回收站，结果符合预期

当有其他用户在编辑文件时，不能将文件移动至回收站，通过多个前端进行测试，结果如下：

```
2021/07/17 14:49:35 recv: {"type":"MTTERROR","data":null}
```

返回错误，测试结果符合预期

Recover

测试恢复文件接口，将文件从回收站中恢复

测试将前面移动到回收站的文件file1恢复

测试前端接受的返回值如下：

```
2021/07/17 14:46:27 recv: {"type":"RECOVERET","data":null}
```

恢复成功，结果符合预期

Clear Trash

测试彻底删除文件接口，将文件从回收站中彻底删除

测试前端接受的返回如下：

```
2021/07/17 14:53:37 recv: {"type":"CTRET","data":null}
```

彻底删除成功，结果符合预期

Rollback

测试回滚文件接口，每次将文件的编辑回滚一步

回滚的返回值有四种：1. 回滚成功 2. 回滚到头 3. 其他人编辑了回滚位置，无权回滚覆盖 4. 其他人持有回滚位置的锁，无法回滚

对上述返回值分别进行测试，测试前端接受的返回值如下：

```
// 正常回滚，返回向回滚终端返回回滚前后的值
2021/07/17 15:04:38 recv: {"type":"ROLLBACK","row":1,"column":1,"newValue":"data1","fileName":"file2","userName":
// 回滚到头，已经不能再回滚了
2021/07/17 15:05:49 recv: {"type":"ROLLBACKEMPTY","data":null}
// 其他用户编辑了对应位置，无法再回滚
2021/07/17 15:08:33 recv: {"type":"ROLLBACKERR","data":null}
// 回滚位置被锁，无法回滚
2021/07/17 15:10:48 recv: {"type":"ROLLBACKLOCK","data":null}
```

四种输出都符合预期

Log

测试输出log接口，直接将log返回到前端

webserver会对创建，打开，编辑，删除，恢复，彻底删除，回滚操作进行记录

前端获取的返回值如下：


```
{"type":"LOG","data":["[CREATE] adam create the file file","[OPEN] adam open the file file","[UPDATE] user:adam s
......
"[ROLLBACK] user:xzl file:file2","[OPEN] xzl open the file file2","[UPDATE] user:xzl set row:1 col:1 from:  to:da
```


data对应log的数组，结果符合预期

测试结果分析

Webserver

在对后端接口进行的功能测试上测试结果都符合预期，没有发现后端的功能存在明显的问题。但是在性能方面存在着一些不足，在执行某些接口时延迟较高，获取webserver返回值的等待时间较长。create接口与open接口需要的时间都比较长，虽然在前端添加了一些加载页面等处理，在使用时的用户体验还是会受到影响。

gfs

可以看到create、write、read的性能基本都是几十到几百毫秒级，还是可以接受的，他们还受到我们多次输出的影响，如果删掉所有的print应该还可以更进一步

改进方法分析

Webserver

尽管功能上没有明显的问题，create与read都存在着优化的可能。对于create，可以先将创建后的文件列表返回前端，降低用户的等待时间，再异步的在文件系统中真正的创建文件。但这就可能出现在没有create完成时用户就进行了对文件的写操作等问题，需要通过对操作进行缓存等方式进行处理。对于open接口，现在的方法是将所有的内容都读出并返回前端，实际上可以只将非空的单元格信息读出并发回给前端，可以通过给每个单元格设置非空符号来实现这一方法，减少传输带宽，优化性能。

gfs

功能上没有明显问题，如果要提高性能，我们可以使用添加cache、删除输出等方法