

分布式文件系统

(1) 功能

1. 在实现文件基本操作的基础上增加更多的文件操作，实现一共10种文件操作，增加文件夹创建、删除、列举所有文件、切换文件目录、重命名、可视化打开文件读取和修改操作等功能。
ls: 列举当前文件夹中的所有文件 cd route: 进入下一级route目录 mkdir fold_name: 创建文件夹 rmdir fold_name: 删除文件夹 rename old_name new_name: 文件夹或文件重命名 mknod file_name: 创建文件 remove file_name: 删除文件 read file_name: 读文件内容 write file_name content: 写文件内容 open file_name: 以可视化形式打开文件，并且可以直接读取和删改
2. 引入两种处理文件方式，一是直接命名行操作，而是可视化操作，可以显示打开文件，方便阅读信息或者直接修改信息。
3. 文件系统命令格式，贴近cmd格式，符合文件系统特点。
4. 加入cd模式，并且考虑当前文件夹位置切换，相应的文件创建、访问和删除也进行相应更新，更加符合实际情况，可用性进一步增强。
5. 自主设计服务器拓扑，不固定一个主服务器，根据实际指定主服务器，更加符合现实的分布式系统。
6. 不局限于一对多单一广播结构，服务器与服务器之间相互进行多播，因此服务器之间的连接方式更加灵活，符合P2P的特点。
7. 增加多种文件操作异常处理，容错性好，保证操作文件时不会出现程序崩溃，合理处理各种不合法操作，增强程序健壮性。
8. 减少单点失效风险，主服务器宕机后，选择次优服务器补上作为主服务器。
9. 服务器支持多线程并发，提高性能，支持多用户同时进行访问。
10. 客户向一个服务器发送rpc请求修改文件数据，服务器使用多线程同时将更新消息多播给其他服务器，其他服务器也采用多线程并发方式进行多播，提高并行效率。
11. 为了保证修改文件数据的读写一致性，使用文件读写锁，在修改文件数据处设立临界区，保证只有一个线程在写，其他线程不能干扰，避免RC问题，读可以多个线程同时进行读取，保证读写一致性。
12. 实现缓存更新，可以加快客户端访问程序的效率。

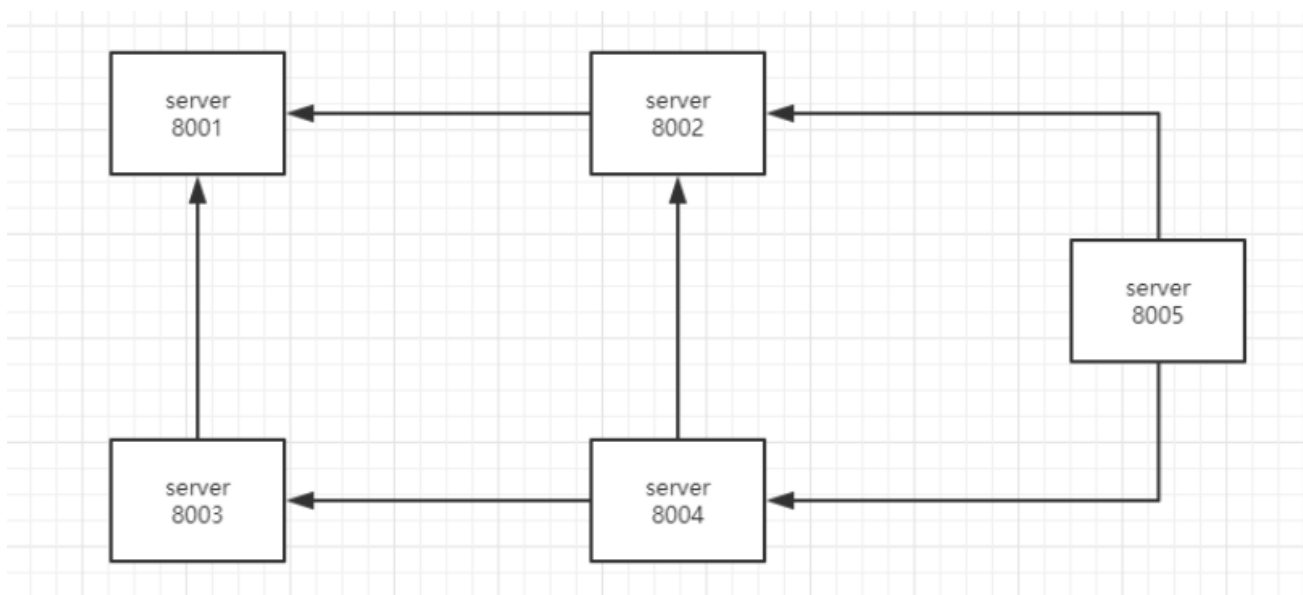
(2) 实现思路

1. 使用python的XML-RPC-Remote Procedure Call框架，即XML远程方法调用,利用http+xml封装进行RPC调用。基于http协议传输、XML作为信息编码格式。一个xml-rpc消息就是一个请求体为xml的http-post请求，服务端执行后也以xml格式编码返回。Xmlrpc的优点有很多，有利于传输复杂的数据，并且通过程序语言的封装，实现远程对象的调用。
2. 本次实验采用一个或多个客户端，5个服务器，自主设立拓扑结构，即5个服务器的连接结构，每个服务器都有一个文件系统副本，相互之间保持一致性。
3. 对于每个服务器，在该服务器定义文件系统操作，包括文件目录创建、删除、列举当前目录的所有文件、切换目录、判断文件类型、创建文件、读取文件数据、写入文件数据、删除文件、以可视化形式操作文件的函数，通过python的OS模块辅助实现，同时针对文件操作遇到的不同情况作异常处理操作，提高容错能力。

4. 客户端访问服务器时，根据指令提示选择执行对应的文件操作，服务器会进行相应的响应。每次客户端想要进行访问服务器时，会根据距离的大小选择最近的服务器X，如果是访问数据，直接访问服务器X，并且获取数据，如果是改写数据，那么先对服务器X中的数据进行修改，然后再让服务器X多播到其他服务器，更新其他服务器中的数据，保持一致性。
5. 如果服务器X宕机，则客户端重新建立连接，选择除了服务器X外最近的服务器建立连接，功能仍然同（3）一样。如果是其他服务器宕机，那么多播时，邻近的服务器不会与其建立连接。
6. 服务器之间的结构并不是一对多，没有指定哪一个服务器一定为主服务器，而是根据客户端访问的位置选择最近的服务器作为主服务器，因此多播时，主服务器不会通过多播就可以发送给所有服务器，而是通过服务器之间相互多播，从而更新整个服务器群体的整个数据，相比一对多的单一结构，本实验结构更加接近实际，同时可靠性更高，避免单一主服务器单点失效的风险。
7. 为了提高并发多播能力，服务器之间多播是采用多线程进行并行传播，相比用户一个个发送rpc请求，显著提高性能。
8. 对文件进行操作时，添加文件读写锁保证读写一致性，避免RC问题。
9. 引入缓存机制，在访问服务器文件数据时，首先查看本地缓存是否存有对应的副本，有则直接读取数据；如果没有，则从服务器下载到本地缓存，再读取数据；在更新文件数据时，如果本地缓存存有副本，则先更新本地缓存的文件，再上传更新到服务器。

(3) 实现细节

1) 程序框图



Server 800是服务器的名称，800是使用的端口，为了测试方便，所有服务器都是在一台电脑进行测试，不同服务器端口号不一样。

如图连接方式，当客户端请求访问时，假设距离客户端最近的服务器为server 8005，则客户端发送一个rpc给server 8005，如果是读取服务器的数据，server 8005直接放回数据给客户端，如果是修改数据，那么server 8005修改数据后，多播给server 8004和8002，通知其对数据进行相应的更新，同时server 8004也会多播给server 8003和server 8002，通知其更新，由于server 8002此时已经更新，因此不再作更新，最后server 8002和server 8003会通知server 8001对数据进行更新，因此最后5个服务器的数据保持一致。

如果server 8005宕机，客户端会选择距离第二近的服务器发送rpc，如果server 8004为第二近，则接收rpc，过程同上。

2) 具体设计

A 服务器

1. 建立服务器：使用多线程服务器，支持多用户同时访问，同时注册类函数，与其他服务器建立连接，开始运行服务器。

```
class Server():

    def __init__(self,port):
        self.port = port
    def build_server(self):
        server = ThreadXMLRPCServer(("localhost", self.port), allow_none=True)
        print ("This is Server ",self.port,"start!")
        self.manage_data = ManageData()
        self.manage_data.store_data()
        self.connect_to_other_server()
        server.register_instance(self.manage_data)
        print()
        server.serve_forever()
```

2. 连接其他服务器：根据拓扑，每个服务器定义好需要连接的服务器端口号，建立好对象连接，存储方便后面多线程使用。

```
def connect_to_other_server(self):
    self.server_port = [8001]
    self.server_proxy = []
    for port in self.server_port:
        proxy = xmlrpc.client.ServerProxy("http://localhost:"+str(port))
        self.server_proxy.append((port,proxy))
        print("Connected to server ",str(port))
        self.manage_data.proxy(self.server_proxy,self.port)
```

定义注册使用的类函数：

3. 创建文件夹：获取当前文件夹的路径，切换到当前服务器放置数据的目录，判断文件夹名字是否已经存在，不存在才创建。

```

def file_mkdir(self, folder_name):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE + '\\' + folder_name
    try:
        os.mkdir(file_path)
    except OSError:
        # print("要创建的文件夹已经存在, 不能重复创建! ")
        return False
    else:
        # print("成功创建新文件夹! ")
        self.update()
        return True

```

4. 删除文件夹：获取目录路径，判断文件夹名字是否存在，是否为空，满足条件才进行删除。

```

def file_rmdir(self, route, folder_name):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if len(route) != 0:
        file_path = file_path + '\\' + route
    if folder_name not in os.listdir(file_path):
        # print("要删除的文件夹不存在, 删除失败! ")
        return 0
    else:
        file_path = file_path + '\\' + folder_name
        try:
            os.rmdir(file_path)
        except OSError:
            # print("当前文件夹不是空的, 不可以删除! ")
            return -1
        else:
            # print("成功删除指定文件夹! ")
            self.update()
            return 1

```

5. 列举当前目录所有的文件：将客户端当前所在的目录下的所有文件全部列举出来。

```

def file_ls(self, file_route):
    file_path = os.getcwd()
    if len(file_route) != 0:
        file_path = file_path + FILE_ROUTE + '\\' + file_route
    else:
        file_path = file_path + FILE_ROUTE
    return os.listdir(file_path)

```

6. 重命名：对文件或文件名字进行重命名，如果旧名字存在或者新名字已经存在，提示命名失败，否则进行命名。

```
def file_rename(self,old_name,new_name,route):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if len(route) != 0:
        file_path = file_path + '\\' + route
    if old_name not in os.listdir(file_path):
        # print("要重命名的文件或文件夹不存在，重命名失败！")
        return 0
    else:
        old_file_path = file_path + '\\' + old_name
        new_file_path = file_path + '\\' + new_name
        try:
            os.rename(old_file_path,new_file_path)
        except OSError:
            # print("新命名的名字已经存在，不能重命名！")
            return -1
        else:
            # print("重命名成功！")
            self.update()
            return 1
```

7. 删除文件：判断文件名字是否存在，文件类型是否正确，满足才进行删除。

```
def file_remove(self,file_name,route):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if len(route) != 0:
        file_path = file_path + '\\' + route
    if file_name not in os.listdir(file_path):
        # print("要删除的文件不存在，删除失败！")
        return 0
    else:
        file_path = file_path + '\\' + file_name
        try:
            os.remove(file_path)
        except OSError:
            # print("删除的为文件夹，删除失败，需要更换另一个指令！")
            return -1
        else:
            # print("删除文件成功")
            self.update()
            return 1
```

8. 创建新文件：如果文件已经存在，创建失败，否则成功执行创建。

```

def file_mknod(self, file_name, route):
    file_path = os.getcwd()
    if len(route) == 0:
        file_path = file_path + FILE_ROUTE + '\\' + file_name
    else:
        file_path = file_path + FILE_ROUTE + '\\' + route + '\\' + file_name
    try:
        file1 = open(file_path, 'w')
        file1.close()
    except:
        # print("要创建的文件已经存在, 创建失败! ")
        return False
    else:
        # print("创建文件成功! ")
        self.update()
        return True

```

9. 切换目录：根据用户输入目录路径判断是否正确，如果目录存在且正确，并且为文件夹类型，则提示用户可以进行切换路径。

```

def file_cd(self, file_route):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE + '\\' + file_route
    try:
        os.listdir(file_path)
    except OSError:
        # print("输入的路径不正确! ")
        return False
    else:
        return True

```

10. 判断是否为文件夹或者文件：使用OS模块的path属性判断是否为文件夹或者文件。

```

def is_dir(self, fold_name):
    file_path = os.getcwd() + FILE_ROUTE + '\\' + folder_name
    return os.path.isdir(file_path)

def is_file(self, file_name):
    file_path = os.getcwd() + FILE_ROUTE + '\\' + file_name
    return os.path.isfile(file_path)

```

11. 读取文件数据：切换目录后，判断文件是否存在，存在则对文件进行上锁，读取数据，解锁，返回数据给用户。

```

def file_read(self, file_name, route):

```

```

file_path = os.getcwd()
file_path = file_path + FILE_ROUTE
if len(route) != 0:
    file_path = file_path + '\\' + route
if file_name not in os.listdir(file_path):
    # print("输入的文件不存在! 读取数据失败")
    return None
else:
    file_path = file_path + '\\' + file_name
    f1 = open(file_path, 'r')
    #文件锁上锁
    self.lock.acquire()
    data = list(f1.readlines())
    #文件锁解锁
    self.lock.release()
    f1.close()
    return data

```

12. 写入数据到文件：切换目录，判断文件是否存在，如果不存在，则写入数据失败，否则对文件进行加锁，写入数据到文件，再解锁，返回反馈给用户。

```

def file_write(self, file_name, content, route):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if len(route) != 0:
        file_path = file_path + '\\' + route
    if file_name not in os.listdir(file_path):
        # print("输入的文件不存在! 写入数据失败")
        return False
    else:
        file_path = file_path + '\\' + file_name
        f1 = open(file_path, 'w')
        #文件锁上锁
        self.lock.acquire()
        f1.writelines(content)
        #文件锁解锁
        self.lock.release()
        # print('数据已经写入文件! ')
        self.update()
        return True

```

13. 以可视化方式操作文件：可以直接打开文件，在该文件直接看到信息或者直接修改信息，程序会一直阻塞，等到用户关闭文件，此时所做的修改会保存。

```

def file_open(self, file_name, route):
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if len(route) != 0:
        file_path = file_path + '\\' + route
    if file_name not in os.listdir(file_path):
        # print("输入的文件不存在! 读取数据失败")
        return 0
    else:
        file_path = file_path + '\\' + file_name
        os.popen(file_path).read()
        self.update()
        return 1

```

14. 多线程多播更新数据：当一个服务器的数据有所更新时，此时采用多线程对邻近的服务器进行多播，传递更新的信息给相邻的服务器，由于采用多线程，因此可以提高并发能力。

```

class MyThread(threading.Thread):
    def __init__(self, threadName, event, port, proxy, manage_data, my_port):
        threading.Thread.__init__(self, name=threadName)
        self.threadEvent = event
        self.port = port
        self.proxy = proxy
        self.manage_data = manage_data
        self.my_port = my_port

    def run(self):
        while True:
            self.threadEvent.wait()
            print('accessing')
            try:
                own_file_path = os.getcwd() + FILE_ROUTE
                self.proxy.file_update(own_file_path, self.my_port)
            except:
                self.threadEvent.clear()
            else:
                self.threadEvent.clear()
                print("Update data to server ", str(self.port))

```

B 客户端

1. 建立连接，根据设计的拓扑选择最短的服务器进行访问，比如当前客户端选择的是服务器8005。


```

class RPC_Client():
    def __init__(self):
        self.ser_mes = [(8001,300),(8002,200),(8003,280),(8004,150),(8005,100)]
        self.ser_mes = sorted(self.ser_mes,key=operator.itemgetter(1))
        self.index = 0
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("This is client,start!")
        print("Currently connect to server ",str(self.current_port))
        self.route = ''
        self.client()

```

2. 指令输入提示：输入help指令，提供指令信息帮助。

```

if len(command) == 1 and command[0] == 'help':
    print('ls: 列举当前文件夹中的所有文件')
    print('cd route: 进入下一级route目录')
    print('mkdir fold_name: 创建文件夹')
    print('rmdir fold_name: 删除文件夹')
    print('rename old_name new_name: 文件夹或文件重命名')
    print('mknod file_name: 创建文件')
    print('remove file_name: 删除文件')
    print('read file_name: 读文件内容')
    print('write file_name content: 写文件内容')
    print('open file_name: 以可视化形式打开文件，并且可以直接读取和删改')

```

3. 列举所有文件，如果当前服务器发生宕机，则自动进行切换服务器，选择第二近的服务器作为连接的服务
器，不会让程序崩溃，下面的指令也是如此，不再进行重复。

```

elif len(command) == 1 and command[0] == 'ls':
    try:
        mes = self.server_proxy.file_ls(self.route)
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        print ("目录为: ",mes)

```

4. 切换目录：首先判断切换的目录是否合法，切换目录后，当前的目录显示也做更改。

```
elif len(command) == 2 and command[0] == 'cd':
    if command[1] == '..':
        if '\\ ' not in self.route and len(self.route) != 0:
            self.route = ''
        else:
            try:
                index1 = self.route.rindex('\\ ')
            except:
                print('输入错误! ')
            else:
                self.route = self.route[0:index1]
    else:
        try:
            mes = self.server_proxy.file_cd(self.route + '\\ ' + command[1])
        except:
            print("server",str(self.current_port),"is down")
            self.index = (self.index+1)%5
            self.current_port = self.ser_mes[self.index][0]
            self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
            print("Reconnect to second server",self.current_port)
            print("Please input the command again")
        else:
            if mes == False:
                print('输入的路径不正确! ')
            else:
                if self.route == '':
                    self.route = command[1]
                else:
                    self.route = self.route + '\\ ' + command[1]
```

5. 创建文件夹：调用创建文件夹函数，给定名字进行创建。

```
elif len(command) == 2 and command[0] == 'mkdir':
    try:
        mes = self.server_proxy.file_mkdir(self.route + '\\ ' + command[1])
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == False:
            print("要创建的文件夹已经存在，不能重复创建! ")
```

```
else:
    print("成功创建新文件夹！")
```

6. 删除文件夹：在合法的情况下删除文件夹。

```
elif len(command) == 2 and command[0] == 'rmdir':
    try:
        mes = self.server_proxy.file_rmdir(self.route, command[1])
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == 0:
            print("要删除的文件夹不存在，删除失败！")
        elif mes == -1:
            print("当前文件夹不是空的,不可以删除！")
        elif mes == 1:
            print("成功删除指定文件夹！")
```

7. 重命名：给文件或文件夹更换名字。

```
elif len(command) == 3 and command[0] == 'rename':
    try:
        mes = self.server_proxy.file_rename(command[1],command[2],self.route)
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == 0:
            print("要重命名的文件或文件夹不存在，重命名失败！")
        elif mes == -1:
            print("新命名的名字已经存在，不能重命名！")
        elif mes == 1:
            print("重命名成功！")
```

8. 创建文件：指定文件名字进行创建。

```

elif len(command) == 2 and command[0] == 'mknod':
    try:
        mes = self.server_proxy.file_mknod(command[1],self.route)
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == False:
            print("要创建的文件已经存在, 创建失败! ")
        elif mes == True:
            print("创建文件成功! ")

```

9. 删除文件：删除指定的文件。

```

elif len(command) == 2 and command[0] == 'remove':
    try:
        mes = self.server_proxy.file_remove(command[1],self.route)
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == 0:
            print("要删除的文件不存在, 删除失败! ")
        elif mes == -1:
            print("删除的为文件夹, 删除失败, 需要更换另一个指令! ")
        elif mes == 1:
            print("删除文件成功")

```

10. 读取文件数据：如果本地缓存有数据，那么直接读取文件数据，否则访问服务器，下载文件数据，读取数据，并且存储文件数据在缓存，供下一次访问。

```

elif len(command) == 2 and command[0] == 'read':
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if command[1] in os.listdir(file_path):
        file_path = file_path + '\\' + command[1]

```

```

        with open(file_path, 'r') as f1:
            data = list(f1.readlines())
        for i in range(len(data)):
            data[i] = data[i][0:-1]
        print('访问的数据已在缓存, 文件数据如下: ')
        print(data)
    else:
        try:
            mes = self.server_proxy.file_read(command[1], self.route)
        except:
            print("server", str(self.current_port), "is down")
            self.index = (self.index+1)%5
            self.current_port = self.ser_mes[self.index][0]
            self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
            print("Reconnect to second server", self.current_port)
            print("Please input the command again")
        else:
            if mes == None:
                print("输入的文件不存在, 读取数据失败!")
            else:
                file_path = file_path + '\\' + command[1]
                with open(file_path, 'w') as f1:
                    f1.writelines(mes)
                for i in range(len(mes)):
                    mes[i] = mes[i][0:-1]
                print("访问数据不在缓存, 从服务器下载, 缓存到本地, 文件数据如下")
                print(mes)

```

11. 写入文件数据: 如果本地缓存有该文件, 那么在本地文件修改后, 再更新到服务器, 否则直接访问服务器, 更新对应文件数据。

```

elif len(command) == 3 and command[0] == 'write':
    file_path = os.getcwd()
    file_path = file_path + FILE_ROUTE
    if command[1] in os.listdir(file_path):
        file_path = file_path + '\\' + command[1]
        with open(file_path, 'w') as f1:
            f1.writelines(command[2])
        print("数据已经更新到本地缓存, 现在上传到服务器中! ")
    try:
        mes = self.server_proxy.file_write(command[1], command[2], self.route)
    except:
        print("server", str(self.current_port), "is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server", self.current_port)
        print("Please input the command again")
    else:

```

```

if mes == False:
    print("输入的文件不存在,写入数据失败!")
elif mes == True:
    print("数据已经更新到服务器文件! ")

```

12. 可视化形式操作文件：直接打开文件，阅读数据或者直接修改，关闭后，修改数据仍然保存。

```

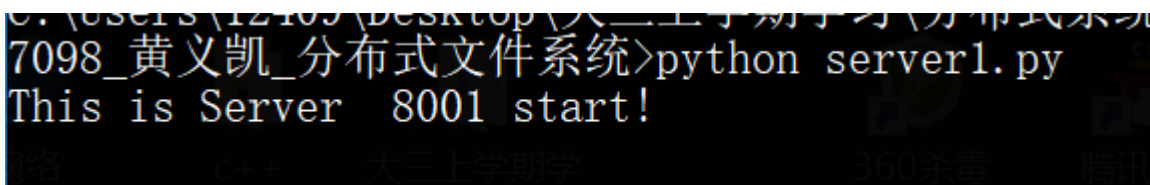
elif len(command) == 2 and command[0] == 'open':
    try:
        mes = self.server_proxy.file_open(command[1],self.route)
    except:
        print("server",str(self.current_port),"is down")
        self.index = (self.index+1)%5
        self.current_port = self.ser_mes[self.index][0]
        self.server_proxy =
xmlrpc.client.ServerProxy("http://localhost:"+str(self.current_port))
        print("Reconnect to second server",self.current_port)
        print("Please input the command again")
    else:
        if mes == 0:
            print("输入的文件不存在,读取数据失败!")
        elif mes == 1:
            print('读取或修改数据完成! ')

```

(4) 程序运行结果

(1) 用cmd打开多个进程，依次运行服务器。

首先打开8001：

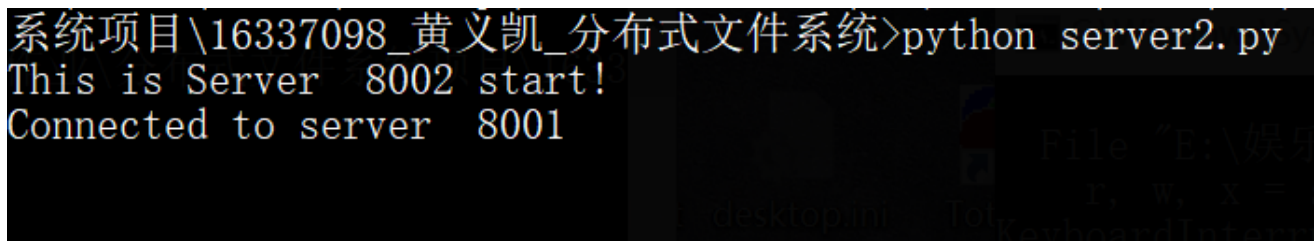


```

C:\Users\12105\Desktop\大三上学期学习\分布式系统
7098_黄义凯_分布式文件系统>python server1.py
This is Server 8001 start!

```

接着打开8002，连接服务器8001：



```

系统项目\16337098_黄义凯_分布式文件系统>python server2.py
This is Server 8002 start!
Connected to server 8001

```

打开服务器8003，连接服务器8001：

```
C:\Users\12109\Desktop\大三上学期学习\分布式系统\作业项目\16337098_黄义凯_分布式文件系统>python server3.py
This is Server 8003 start!
Connected to server 8001
```

打开8004, 连接8002,8003:

```
目\16337098_黄义凯_分布式文件系统>python server4.py
This is Server 8004 start!
Connected to server 8002
Connected to server 8003
```

最后打开8005, 连接8002, 8004:

```
统项目\16337098_黄义凯_分布式文件系统>python server5.py
This is Server 8005 start!
Connected to server 8002
Connected to server 8004
```

- 启动客户端, 为了模拟实验, 自定义客户端到5个服务器的距离, 然后选择最短距离服务器优先进行访问。

```
self.ser_mes = [(8001,300),(8002,200),(8003,280),(8004,150),(8005,100)]
self.ser_mes = sorted(self.ser_mes,key=operator.itemgetter(1))
```

- 在本设计中, 最短距离的是server 8005, 因此选择server 8005建立连接。

```
C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业项目\16337098_黄义凯_分布式文件系统>python client.py
This is client,start!
Currently connect to server 8005
-----
输入help可以得到更多的指令!
>
```

- 多个进程同时运行

```
C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业\分布式文件系统>python server1.py
This is Server 8001 start!

C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业\分布式文件系统>python server2.py
This is Server 8002 start!
Connected to server 8001

C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业\分布式文件系统>python server3.py
This is Server 8003 start!
Connected to server 8001

C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业\分布式文件系统>python server4.py
File "server4.py", line 256, in build_server
server.serve_forever()
File "E:\娱乐软件\python\lib\socketserver.py", line 226, in serve_forever
ready = selector.select(poll_interval)
File "E:\娱乐软件\python\lib\selectors.py", line 314, in select
r, w, x = self.select(r, w, w, timeout)
KeyboardInterrupt

C:\Users\12409\Desktop\大三上学期学习\分布式系统\作业\分布式文件系统>python client.py
This is client, start!
Currently connect to server 8005
输入help可以得到更多的指令!
```

- 输入help获取指令说明

```
>help
ls: 列举当前文件夹中的所有文件
cd route: 进入下一级route目录
mkdir fold_name: 创建文件夹
rmdir fold_name: 删除文件夹
rename old_name new_name: 文件夹或文件重命名
mknod file_name: 创建文件
remove file_name: 删除文件
read file_name: 读文件内容
write file_name content: 写文件内容
open file_name: 以可视化形式打开文件，并且可以直接读取和删改
```

- (2) 列举当前目录中所有的文件。

```
>ls
目录为: ['file', 'file1', 'test.txt']
```

- (3) 创建文件夹

1. 如果文件夹已经存在，创建失败。

```
>mkdir file1
要创建的文件夹已经存在，不能重复创建!
```

2. 否则创建文件夹成功。


```
>mkdir file2
成功创建新文件夹!
```

查看文件夹，可知file2文件夹成功创建。

名称	修改日期	类型	大小
file	2019/1/2 20:20	文件夹	
file1	2019/1/2 20:21	文件夹	
file2	2019/1/2 23:35	文件夹	
test.txt	2019/1/2 20:22	文本文档	1 KB

(4) 删除文件夹

1. 如果要删除的文件夹不存在，删除失败。

```
>rmdir file3
要删除的文件夹不存在，删除失败!
```

2. 要删除的文件夹不是空的，删除失败。

```
>rmdir file
当前文件夹不是空的, 不可以删除!
```

3. 成功删除文件夹。

```
>rmdir file2
成功删除指定文件夹!
```

此时，已经没有该文件夹：

file	2019/1/2 23:44	文件夹	
file1	2019/1/2 20:21	文件夹	
test.txt	2019/1/2 20:22	文本文档	1 KB

(5) 重命名

1. 要重命名的文件或文件夹不存在，重命名失败。

```
>rename file3 file4
要重命名的文件或文件夹不存在，重命名失败!
```

2. 新命名的名字已经存在，不能重命名。

```
>rename file1 file
新命名的名字已经存在，不能重命名!
```

3. 重命名成功。

```
>rename file1 file2
重命名成功!
```

经过验证，名字已经更改。

file	2019/1/2 23:44	文件夹	
file2	2019/1/2 20:21	文件夹	
test.txt	2019/1/2 20:22	文本文档	1 KB

(6) 创建文件

1. 创建文件成功。

```
>mknod test1.txt
创建文件成功!
```

查看，存在test1.txt文件。

file	2019/1/2 23:44	文件夹	
file2	2019/1/2 20:21	文件夹	
test.txt	2019/1/2 23:49	文本文档	0 KB
test1.txt	2019/1/2 23:50	文本文档	0 KB

(7) 删除文件

1. 要删除的文件不存在。

```
>remove test2.txt
要删除的文件不存在，删除失败!
```

2. 删除类型不对。

```
输入help可以得到更多的指令！
>remove file
删除的为文件夹，删除失败，需要更换另一个指令！
```

3. 成功删除文件。

```
输入help可以得到更多的指令！
>remove test1.txt
删除文件成功
```

再次查看，该文件已经不存在：

file	2019/1/2 23:44	文件夹	
file2	2019/1/2 20:21	文件夹	
test.txt	2019/1/2 23:49	文本文档	0 KB

(8) 切换目录

1. 进入下一级目录。

```
输入help可以得到更多的指令！
>cd file

-----

输入help可以得到更多的指令！
file>cd file2

-----

输入help可以得到更多的指令！
file\file2>
```

2. 返回上一级目录。

```

file\file2>cd ..
-----
输入help可以得到更多的指令!
file>cd ..
的目录。
-----
输入help可以得到更多的指令!
>

```

3. 在不同的目录对文件的操作也相应更新到对应的目录。

```

file\file2>mknod test3.txt
创建文件成功!

```

此时，在file/file2目录下有该文件夹：

电脑 > 桌面 > 大三上学期学习 > 分布式系统 > 作业 > 分布式文件系统项目 > 16337098_黄义凯_分布式文件系统 > file_data > server > server5 > file > file2			
名称	修改日期	类型	大小
 test3.txt	2019/1/2 23:56	文本文档	0 KB

(9) 读取文件数据

1. 如果在本地已经缓存数据，直接访问。

```

>read test.txt
访问的数据已在缓存，文件数据如下：
['1234', '第十届', 'lihua']

```

2. 输入的文件名字在服务器的文件目录不存在，读取失败。

```

>read test1.txt
输入的文件不存在, 读取数据失败!

```

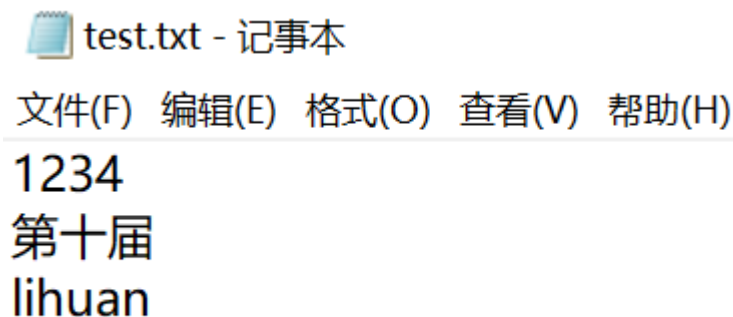
3. 从服务器下载文件数据，缓存到本地，获取数据。

```

>read test.txt
访问数据不在缓存，从服务器下载，缓存到本地，文件数据如下
['1234', '第十届', 'lihua']

```

打开文件夹查看，验证正确。



(10) 写入文件数据

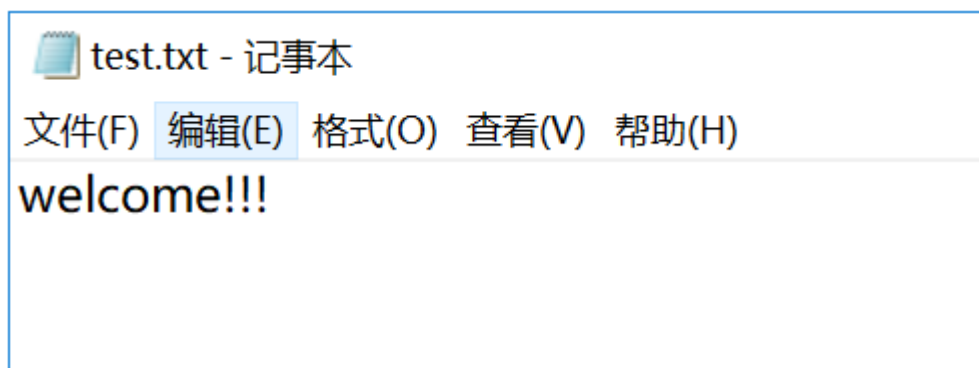
1. 如果本地缓存中有对应的文件，则写入本地文件数据，更新上传到服务器。

```
>write test.txt welcome!!!  
数据已经更新到本地缓存，现在上传到服务器中！  
数据已经更新到服务器文件！
```

2. 输入的文件名字在服务器的文件目录不存在，写入失败。

```
>write test1.txt welcome!!!  
输入的文件不存在, 写入数据失败！
```

3. 数据更新到服务器。



4. 同时服务器进行相互多播，更新其他服务器的数据，保持一致性。

- 8005多播给8004和8002：

```
127.0.0.1 - - [03/Jan/2019 00:00:28] "POST /RPC2 HTTP/1.1" 200 -  
accessing  
Update data to server 8004  
Update data to server 8002  
127.0.0.1 - - [03/Jan/2019 00:01:56] "POST /RPC2 HTTP/1.1" 200 -  
腾讯云 - database.txt
```

- 8004接收8005，更新后多播给8003、8002：

```
127.0.0.1 - - [03/Jan/2019 00:00:29] "POST /RPC2 HTTP/1.1" 200 -  
accessing  
accessing  
Update data to server 8003  
Update data to server 8002
```

- 8003接收到8004的更新信息，更新后，多播给8001：

```
Update data from server 8004  
127.0.0.1 - - [03/Jan/2019 00:11:44] "POST /RPC2 HTTP/1.1" 200 -  
accessing  
Update data to server 8001
```

- 8002接收8005/8004更新请求，更新再传给8001：

```
Update data from server 8005  
127.0.0.1 - - [03/Jan/2019 00:11:43] "POST /RPC2 HTTP/1.1" 200 -  
accessing  
Update data from server 8004  
Update data to server 8001
```

- 8001接收更新请求，进行更新。

```
Update data from server 8003  
Update data from server 8002  
127.0.0.1 - - [03/Jan/2019 00:11:45] "POST /RPC2 HTTP/1.1" 200 -  
127.0.0.1 - - [03/Jan/2019 00:11:45] "POST /RPC2 HTTP/1.1" 200 -
```

(11) 可视化文件操作

- 操作同上，也是会通过多播保证一致性。

```
输入help可以得到更多的指令！  
>open test.txt
```

```
test.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
welcome!!!
```

(12) 宕机测试

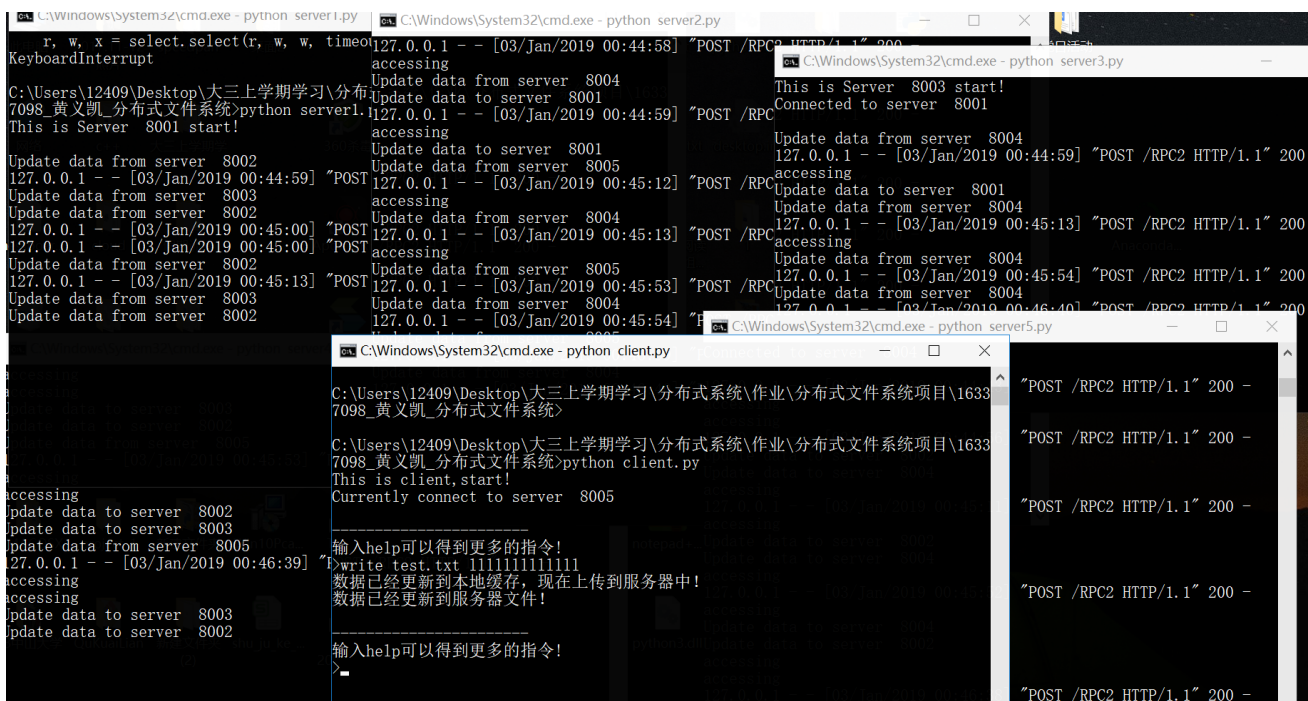
1. 假设服务器8005宕机，此时根据拓扑，服务器选择距离第二近的服务器8004与客户端交互，8004宕机后，依次类推。

```
>write test.txt huhuhu
数据已经更新到本地缓存，现在上传到服务器中！
server 8005 is down
Reconnect to second server 8004
Please input the command again

-----
输入help可以得到更多的指令！
>
```

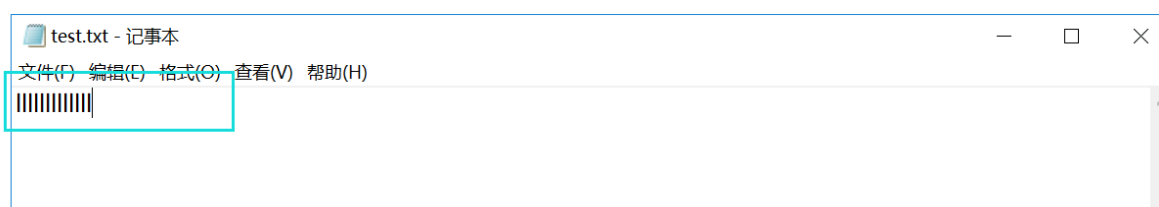
2. 修改数据仍然能保持一致性。

- 多播更新过程同 (10)



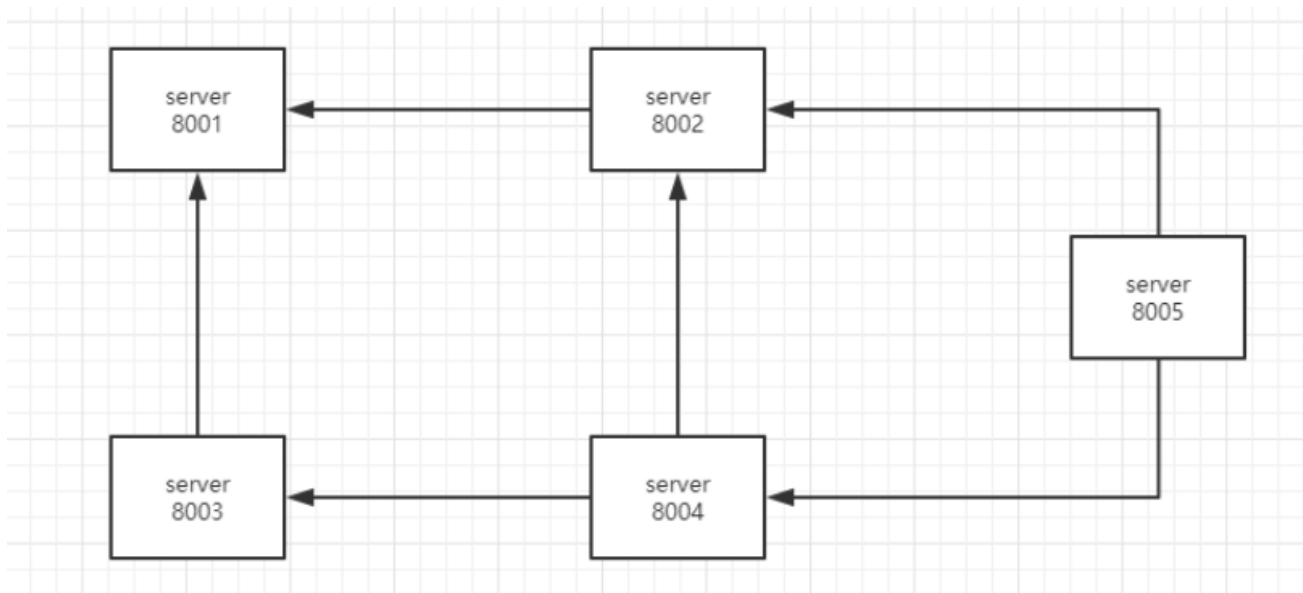
- 通过比较，可得server1的文件信息与客户端输入的数据信息一致，说明一致性得以保证。

桌面 > 大三上学期学习 > 分布式系统 > 作业 > 分布式文件系统项目 > 16337098_黄义凯_分布式文件系统 > file_data > server > server1			
名称	修改日期	类型	大小
file2	2019/1/3 0:45	文件夹	
test.txt	2019/1/3 0:45	文本文档	1 KB



(5) 总结

1. 在本次项目中，我对RPC的使用有进一步的熟悉，对远程过程调用的原理有一个更深刻的理解，通过服务器设计，一开始我是指定一个服务器为主服务器，主要负责跟客户端交互，其余服务器跟主服务器是多对一的结构，但是这样的结构不合理，没有很好地提高性能，同时如果主服务器宕机，那整个服务器结构就崩掉了，单点失效的风险太大，实用性不高，同时多对一结构也不符合实际，性能反而没有提升，因此经过分析，决定放弃多对一的结构，进而设计其他拓扑，如图，每个服务器没有直接与所有服务器进行连接，而是通过其他服务器间接连接，那么多播时，服务器接收到更新请求时，又会将请求发给其他相邻的服务器，从而实现多播更新，并且根据距离远近选择其中一个服务器为主服务器与客户端交互，如果主服务器宕机，那么选择第二近作为替补补上主服务器，有效提高服务器之间的可靠性。



2. 客户端与服务端建立连接后，服务器之间也是需要连接的，但是xmlrpc跟socket不太一样，不会一直保持连接，因此为了能够让服务器之间能够传递信息，根据拓扑图有选择地让服务器与另一个服务器建立单向连接，在传播更新时，服务器沿着单向的路径传播给其他服务器进行更新，同时避免另一个服务器重复往相反方向传播更新信息，有效保证所有服务器都进行更新。
3. 本次实验对文件系统的底层机制有进一步的认识，熟悉了文件创建、访问、删除、列举所有文件，切换目录，创建文件夹、删除文件夹，对文件进行读写操作等功能，将文件机制的实现与RPC远程过程调用结合封装起来，在服务器注册运行，留下接口给用户进行调用，为了能够友好访问，用户也需要设计好访问接口，方便交互，从而实现文件系统的操作。
4. 服务器应该具有一定的容错能力，本次实验自主设计连接拓扑，自定义服务器距离，根据实际距离远近，客户端选择最近的服务器建立连接，如果当前服务器宕机，应该要保证一个点失效不会导致整个服务器群崩溃，继而选择距离第二近的服务器继续建立连接访问，保证容错能力。
5. 使用多个服务器时，应该复制多个副本，更新到各个服务器上，保持一致性，而每次对其中一个服务器更新时，此时服务器之间可以进行多播，传递更新消息，各个服务器接收到更新消息时更新数据，并且同时多播给相邻的其他服务器，这个过程可以使用多线程并发实现，提高并行能力。

6. 对文件进行操作时，如果是读取文件，应该加共享锁，别的用户进行访问时，如果也是读取文件，此时可以访问，如果是写文件，则被拒绝；如果是写入文件，应该加排他锁，此时无论别的用户读还是写文件，一概拒绝，直到该用户使用完释放锁；在分布式系统中，可以考虑另一种情况，为了全局统一，当一个用户对一个服务器的一个文件进行操作时，此时可以在整个服务器群对该文件进行加锁，这可以通过广播一条更新信息实现。
7. 缓存是一个提高访问效率的方法，使用缓存有一个注意的地方，如果服务器上的数据更新，缓存没有更新，那么此时用户访问得到的是旧的数据，不符合实际，解决这个问题有两个方法：一个是缓存定期清空或者定期更新，这样也会有可能访问旧的数据，但是频率会降低，而且不会每次访问都要下载全部数据；二是给文件指定一个标号，每次访问只发送一个标号，跟服务器文件的标号进行比对，如果服务器文件有更新，则标号也会有更新，此时直接把更新的部分数据传送下来，如果没有更新，则标号一样，此时就可以直接访问缓存，这样的好处就是不会有版本不一致的情况，并且可以减少传送量。