带 GUI 客户端的 XML-RPC 远程文件共享

作　　者：　刘奕辰

指　　导：　陈建文老师

目　　录

摘要：

本文介绍了 XML-RPC 远程文件共享及其 GUI 客户端的设计。本程序是一个 P2P 的文件共享程序，通过在本主机上建立一个由 3 个节点组成的虚拟网络，实现 P2P 交互，进行文件传输。本文介绍了对此项目的分析、设计、编码、测试的过程，并提供了全部源代码。

关键字：

软件工程，文件共享，XML-RPC，P2P，GUI。

一. 文件共享概述

1. XML-RPC

xml-rpc 的全称是 XML Remote Procedure Call，即 XML 远程方法调用。它是一套基于 Internet 过程调用而实现了平台无关性与语言无关性的标准规范[1]。

XML-RPC 是通过 HTTP 传输 XML 来实现远程过程调用的 RPC，因为是基于 HTTP、并且使用 XML 文本的方式传输命令和数据，所以兼容性更好，能够跨域不同的操作系统、不同的编程语言进行远程过程调用，凡有所得，必有所失，在兼容性好的同时速度也会慢下来[2]。XML-RPC 的优点是简单、轻量，可读性强，支持多语言多平台。

XmlRPC 客户端工作原理：Client 根据指定 URL 找到服务端地址，然后编码请求数据，调用服务端上的指定服务的方法，接收到服务端的返回,解析响应包，拿出调用的返回结果。

XmlRPC 服务端工作原理：启动一个服务程序， 注册每个能提供的服务，每个服务对应一个 Handler 类，进入服务监听状态，等待 Client 的请求。

2. P2P

点对点技术又称对等互联网络技术，是一种网络新技术，依赖网络中参与者的计算能力和带宽，而不是把依赖都聚集在较少的几台服务器上。纯点对点网络没有客户端或服务器的概念，只有平等的同级节点，同时对网络上的其它节点充当客户端和服务器。这种网络

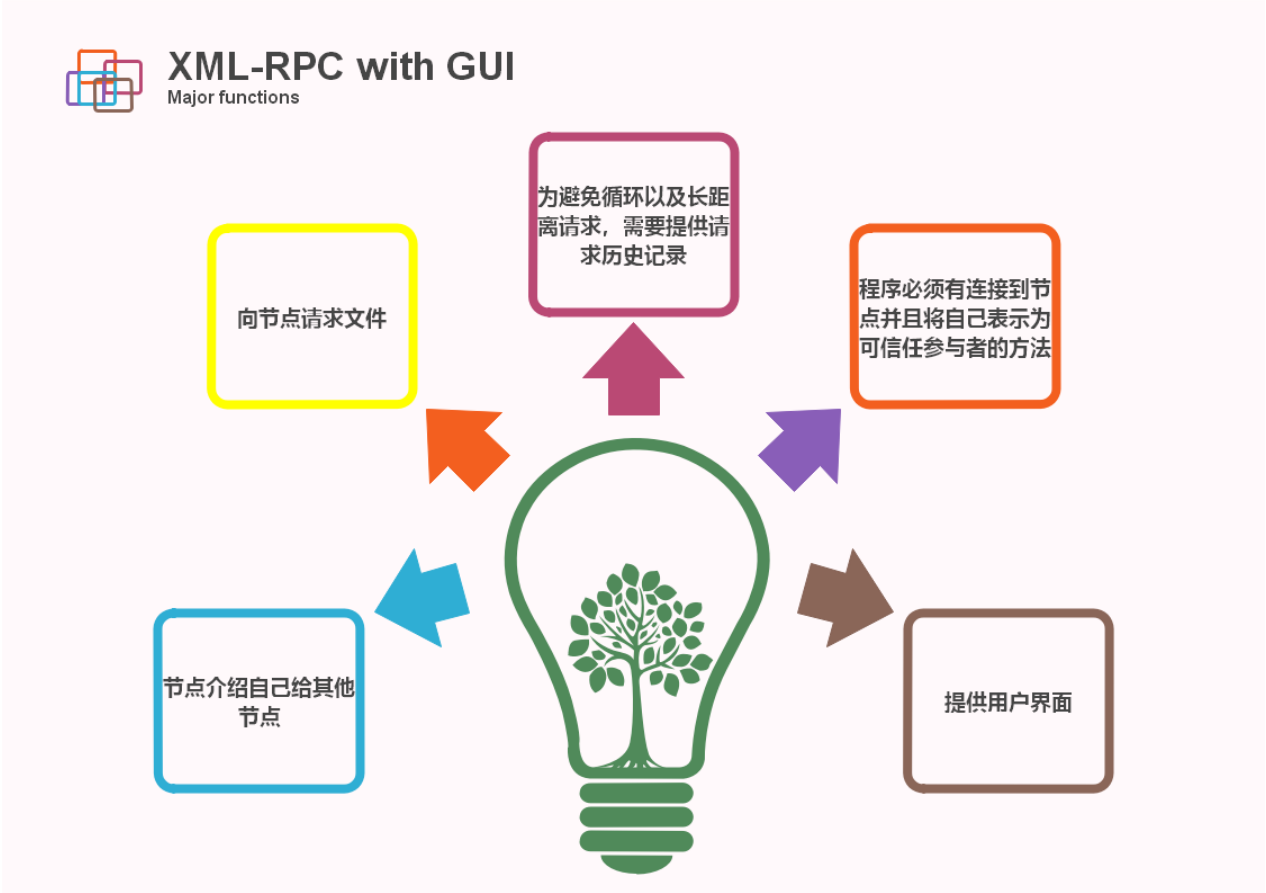
设计模型不同于客户端-服务器模型，在客户端-服务器模型中通信通常来往于一个中央服务器[3]。

二．项目概述

本项目需要创建一个基于 XML-RPC 的 P2P 文件共享程序。

文件共享意味着在运行在不同计算机上的程序之间交换文件，包括文本文件、音频文件等等。在本程序的 P2P 交互中，任何节点（peer）都可以链接到其他节点。

本程序需要实现的主要功能：



三．项目设计工具

在本程序中，我们使用 Pycharm+Anaconda 作为开发环境。

本程序需要用到的主要工具有：

1. 模块 `xmlrpc`

   xmlrpc 是一个集合了 XML-RPC 服务端与客户端实现模块的包。这些模块是: `xmlrpc.client` 和 `xmlrpc.server`.
   xmlrpc 的使用十分简单，只需用服务器的 URL 创建一个 ServerProxy 对象，之后马上就可以访问远程过程。
   其中 `xmlrpc.server` 包含 SimpleXMLRPCServer，它使用（服务器，端口）形式的元组进行实例化。服务器名就是服务器在其上运行的计算机名称（可以使用空字符串表示使用 localhost，也就是正在执行程序的计算机）。

2. 模块 `threading`

   用于实现本程序的并行效果。

3. 模块 `urlparse`

   用于从 url 中提取成分。

4. 其他模块：

   `random、string、time、os.path.`

四. 项目设计步骤

1. 需求分析

对于用户来说，在两台或多台不同的主机上进行文件传输是每天都要用到的技术。在互联网飞速发展的今天，每个用户待共享的文件量，以及用户总数量都在迅速上升。因此，我们需要认识到，为两台及多台终端提供远程文件传输服务的重要性。同时，为了最大程度上满足用户的需求，我们要设计简洁、可视化的用户操作界面，以求得最优的用户体验。

我们分别从**文件共享**和 **GUI 客户端**这两个方面来分析需求。

1. 文件共享需求分析

- 每个节点必须跟踪记录一个已知节点的集合，从而可以向这些节点寻求帮助。节点必须可以将自己“介绍”给其他节点（这样就可以把自身包括在节点集合内部）。
- 必须可以向节点请求文件（通过提供文件名）。如果节点拥有这个文件，那么将其返回；否则它应该轮流询问自己的各个相邻节点，请求相同的文件（之后这些节点又会轮流请求自己的相邻节点）。如果其中的某个节点拥有文件，那么就返回该文件。
- 为了避免循环（A 请求 B，B 又请求 A）以及相邻结点非常长的请求链（A 请求 B 请求 C...再请求 Z），程序必须在请求节点的时候提供请求历史记录。这个历史记录只是一个列表，其中包括到这个节点为止已经参与这次请求的节点。通过不询问在历史记录中存在的节点，就可以避免循环，限制历史列表的长度，则可以避免过长的请求链。
- 程序必须有连接到节点并且将自己标识为可信任参与者的方法。这样就获得了对于不信任参与者（比如在 P2P 网络内的

其他节点）不可用的访问功能。这些功能包括要求节点（通过查询请求）从网络中的其他节点处下载并存储文件。

2. GUI 客户端需求分析

- 应允许客户输入文件名，并将其提交给服务端的 fetch 方法。
- 应该能够列出服务器文件目录中当前可用的文件（包括原始文件和后来下载的文件）。

2. 设计

将设计分为两个阶段：**总体设计阶段**与**详细设计阶段**。

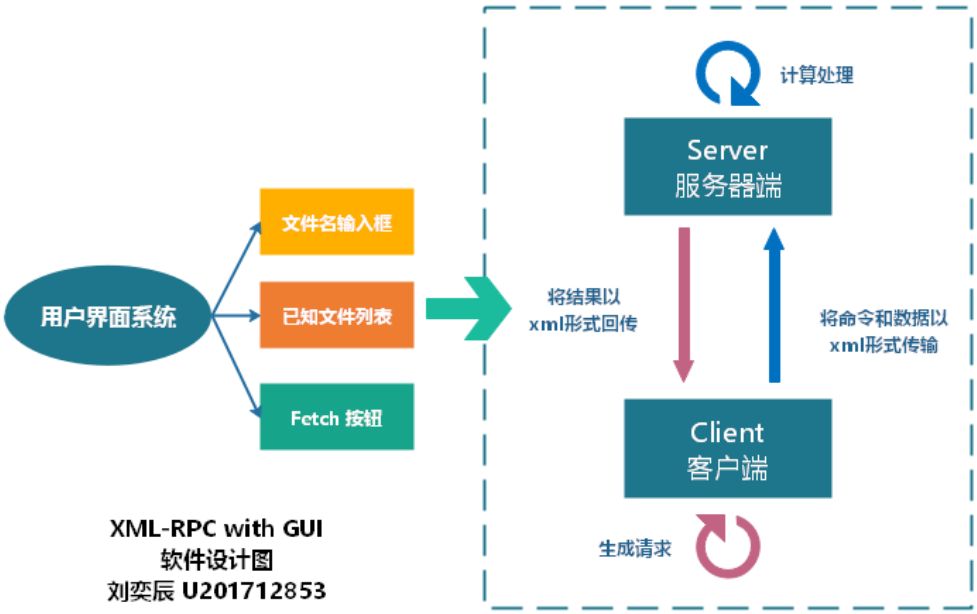
1. 总体设计

在总体设计阶段，首先确定了环境，即操作系统以及编译器。这样，就可以以需求分析说明书为依据，针对环境进行有针对性的设计。

- 操作系统：Windows 10
- 编译器：Pycharm
- 环境：Conda

2. 详细设计

在详细设计阶段，我们将根据用户需求和软件功能将系统划分成三个子系统（Server 子系统、Client 子系统、用户界面子系统），并以框图的形式展现各个子系统之间的功能及其联系。



XML-RPC with GUI
软件设计图
刘奕辰 U201712853

在本程序中，为了实现 P2P，**我们在主机电脑上启动了三个不同的节点，端口号分别为 localhost:6741，localhost:6742，localhost:6743，且分别分配三个文件夹，files1，files2，files3，用于进行三点文件传输及文件共享。**具体见下图：

3. 编码

现将全部代码展示如下：

1. Client.py

```
from cmd import Cmd
from os import path
from random import choice
from string import ascii_lowercase
from server import Node, UNHANDLED
from threading import Thread
from time import sleep
from xmlrpc.client import ServerProxy, Fault
import sys

HEAD_START = 0.1  # Seconds
SECRET_LENGTH = 100
```

```
def randomString(length):
    chars = []
    letters = ascii_lowercase[:26]
    while length > 0:
        length -= 1
        chars.append(choice(letters))
    return ''.join(chars)


class Client(Cmd):
    prompt = '> '

    def __init__(self, url, dirname, urlfile):
        Cmd.__init__(self)
        self.secret = randomString(SECRET_LENGTH)
        n = Node(url, dirname, self.secret)
        t = Thread(target=n._start)
        t.setDaemon(1)
        t.start()

        sleep(HEAD_START)
        self.server = ServerProxy(url)
        urlfile = path.join(dirname, urlfile)
        for line in open(urlfile):
            line = line.strip()
            self.server.hello(line)

    def do_fetch(self, arg):
        try:
            self.server.fetch(arg, self.secret)
        except Fault as f:
            if f.faultCode != UNHANDLED: raise
            print("Couldn't find the file", arg)
```

```
    def do_exit(self, arg):
        print()
        sys.exit()

    do_EOF = do_exit


def main():
    urlfile, directory, url = sys.argv[1:]
    client = Client(url, directory, urlfile)
    client.cmdloop()


if __name__ == '__main__': main()
```

2. Server.py

```
from os.path import join, isfile, abspath
from xmlrpc.client import Fault
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.client import ServerProxy
from urllib.parse import urlparse
import sys

MAX_HISTORY_LENGTH = 6

UNHANDLED = 100
ACCESS_DENIED = 200


class UnhandleQuery(Fault):
#无法处理的查询异常情况
    def __init__(self, message="Couldn't handle the
query"):
        Fault.__init__(self, UNHANDLED, message)
```

```
class AccessDenied(Fault):
#用户访问未被授权的资源时出现的异常情况
    def __init__(self, message="Acces denied"):
        Fault.__init__(self, ACCESS_DENIED,
message)


def inside(dir, name):
#检查给定目录中是否有给定的文件名
    dir = abspath(dir)
    name = abspath(name)
    return name.startswith(join(dir, ''))


def getPort(url):
#从 url 中提取端口号
    name = urlparse(url)[1]
    parts = name.split(':')
    return int(parts[-1])


class Node:
#P2P 网络中的节点。

    def __init__(self, url, dirname, secret):
        self.url = url
        self.dirname = dirname
        self.secret = secret
        self.known = set()

    def query(self, query, history=[]):
        """
        查询文件，可能会向其他已知节点请求帮助。讲文件
```

```
作为字符串返回。
        """
        print('in function query')
        try:
            content = self._handle(query)
        except:
            history = history + [self.url]
            print('in funciton query,
history : %s' % history)
            if len(history) >= MAX_HISTORY_LENGTH:
raise
            content = self._broadcast(query,
history)
        return content

    def hello(self, other):
        """
        用于将节点介绍给其他节点。
        """
        self.known.add(other)
        return 0

    def fetch(self, query, secret):
        """
        用于让节点找到文件并且下载。
        """
        print(query)
        print(secret)
        if secret != self.secret: raise
AccessDenied
        result = self.query(query)
        print(result)
        f = open(join(self.dirname, query), 'w')
        f.write(result)
        f.close()
```

```
        return 0

    def _start(self):
        """
        内部使用，用于启动 XML_RPC 服务器。
        """
        s = SimpleXMLRPCServer(("",
getPort(self.url)))
        s.register_instance(self)
        s.serve_forever()

    def _handle(self, query):
        """
        内部使用，用于处理请求。
        """
        dir = self.dirname
        name = join(dir, query)
        print('in function _handle, file
name : %s' % name)
        if not isfile(name): raise UnhandleQuery
        if not inside(dir, name): raise
AccessDenied
        return open(name).read()

    def _broadcast(self, query, history):
        for other in self.known.copy():
            print('other: %s' % other)
            if other in history: continue
            try:
                s = ServerProxy(other)
                return s.query(query, history)

            except Fault as f:
                if f.faultCode == UNHANDLED:
                    pass
```

```
                else:
                    self.known.remove(other)
            except:
                self.known.remove(other)
        raise UnhandleQuery


def main():
    url, directory, secret = sys.argv[1:]
    n = Node(url, directory, secret)
    n._start()


if __name__ == '__main__': main()
```

3. guic.py (带 gui 的 xml-rpc 远程传输程序)

```
from xmlrpc.client import ServerProxy, Fault
from server import Node, UNHANDLED  # 引入前面的程序
from client import randomString  # 引入前面的程序
from threading import Thread
from time import sleep
from os import listdir
import sys
import wx


HEAD_START = 0.1  # Seconds
SECRET_LENGTH = 100

app = wx.App()

class ListableNode(Node):
    def list(self):
        return listdir(self.dirname)
```

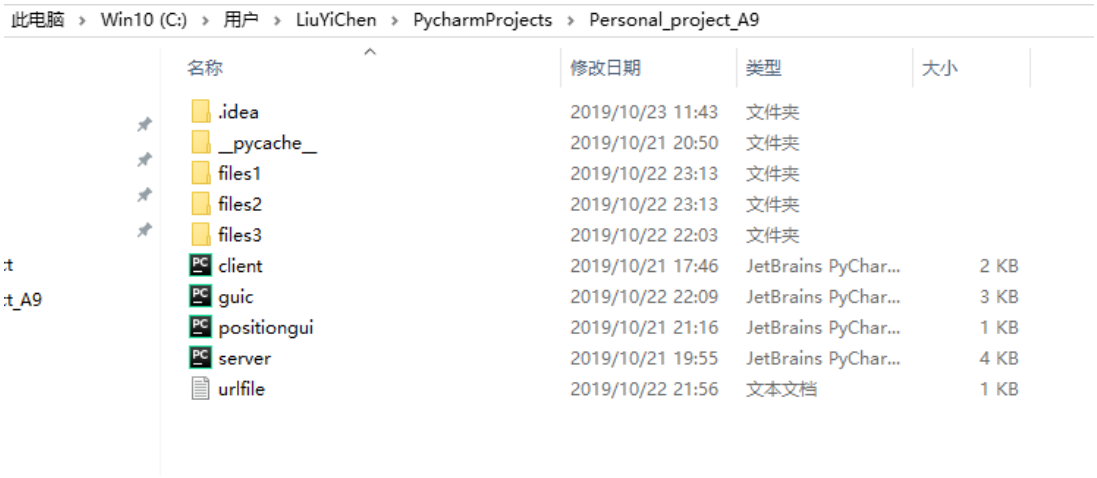
```
class Client(wx.App):
    def __init__(self, url, dirname, urlfile):
        self.secret = randomString(SECRET_LENGTH)
        n = ListableNode(url, dirname, self.secret)
        t = Thread(target=n._start)
        t.setDaemon(1)
        t.start()

        sleep(HEAD_START)
        self.server = ServerProxy(url)
        for line in open(urlfile):
            line = line.strip()
            self.server.hello(line)

        # run gui
        super(Client, self).__init__()

    def updateList(self):
        self.files.Set(self.server.list())

    def OnInit(self):
        win = wx.Frame(None, title="Jerilyn Liu's
File Sharing Client", size=(400, 399))

        bkg = wx.Panel(win)

        self.input = input = wx.TextCtrl(bkg)

        submit = wx.Button(bkg, label="Fetch",
size=(80, 25))
        submit.Bind(wx.EVT_BUTTON,
self.fetchHandler)

        hbox = wx.BoxSizer()
```

```
        hbox.Add(input, proportion=1, flag=wx.ALL |
wx.EXPAND, border=10)
        hbox.Add(submit, flag=wx.TOP | wx.BOTTOM |
wx.RIGHT, border=10)

        self.files = files = wx.ListBox(bkg)
        self.updateList()

        vbox = wx.BoxSizer(wx.VERTICAL)
        vbox.Add(hbox, proportion=0,
flag=wx.EXPAND)
        vbox.Add(files, proportion=1,
flag=wx.EXPAND | wx.LEFT | wx.RIGHT | wx.BOTTOM,
border=10)

        bkg.SetSizer(vbox)

        win.Show()

        return True

    def fetchHandler(self, event):
            query = self.input.GetValue()
            try:
                self.server.fetch(query,
self.secret)
                self.updateList()
            except Fault as f:
                if f.faultCode != UNHANDLED: raise
                print ("couldn't find the file ",
query)
```

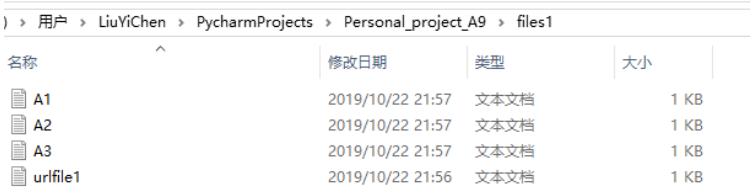
```
def main():
    urlfile, directory, url = sys.argv[1:]
    client = Client(url, directory, urlfile)
    client.MainLoop()


if __name__ == '__main__':
    main()
```

4. 测试（运行结果）

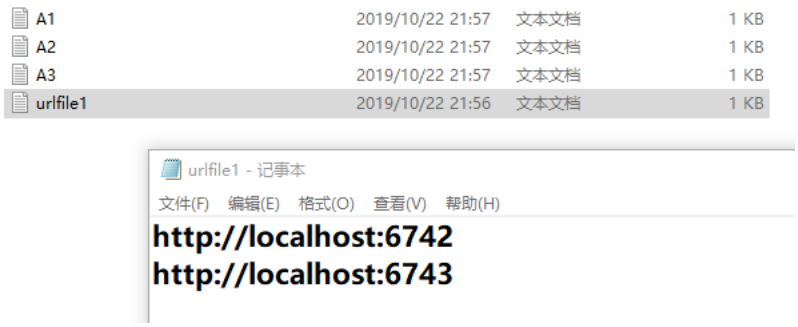
首先，我们在本机程序目录下建立三个文件夹 files1、files2、files3。



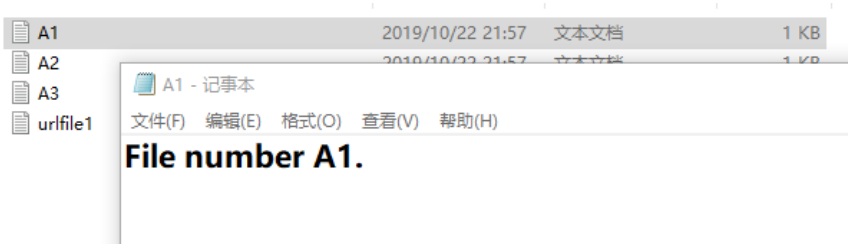
在文件夹 files1 里，放入文件 urlfile1.txt , A1.txt, A2.txt, A3.txt.



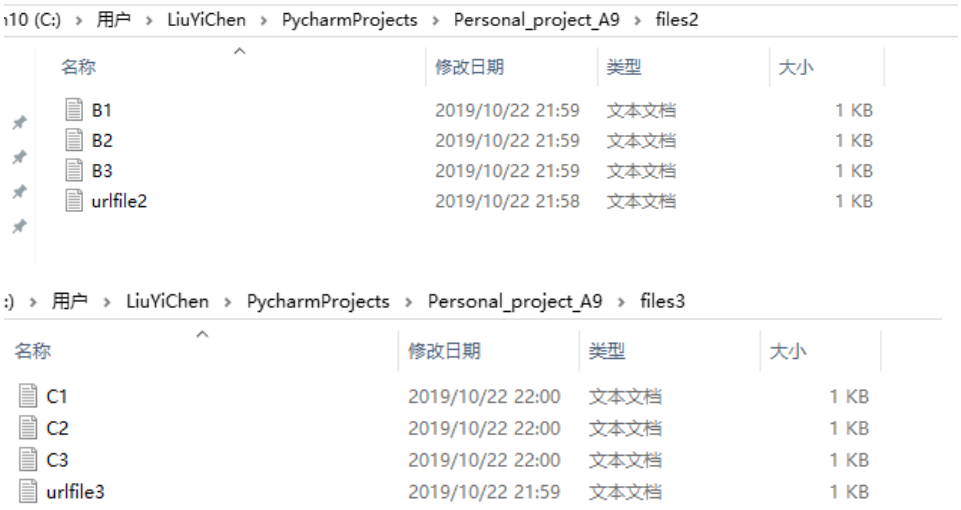
其中 urlfile1.txt 里存入第二个，第三个节点的 url。



A1.txt 等作为测试文件：



其余两个 files2、files3 与 files1 同理，放入测试文件及其他两节点的 url。

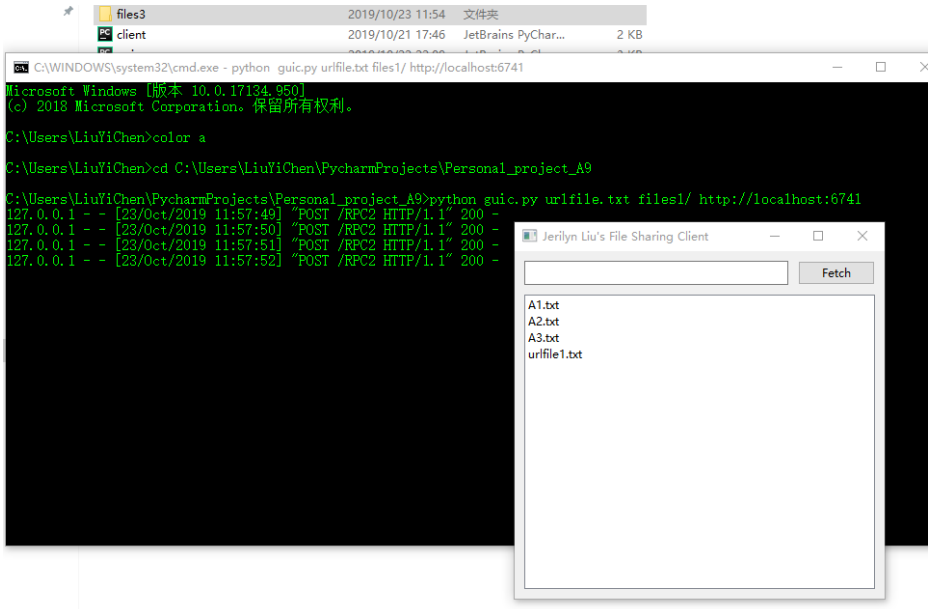


至此，三个节点的文件已经放置好，下面我们通过 cmd 启动三个节点。先 cd 到程序目录下，再开启 guic.py 文件，同时我们需要提供一个 urlfile.txt,里面含有所有节点的 url。再者，要提供一个用于共享文件的文件夹名，以及待启动节点的 url。

1. 启动节点 http://localhost:6741

-Cd C:\Users\LiuYiChen\PycharmProjects\Personal_project_A9

- python guic.py urlfile.txt files1/ http://localhost:6741 #启动第一个节点

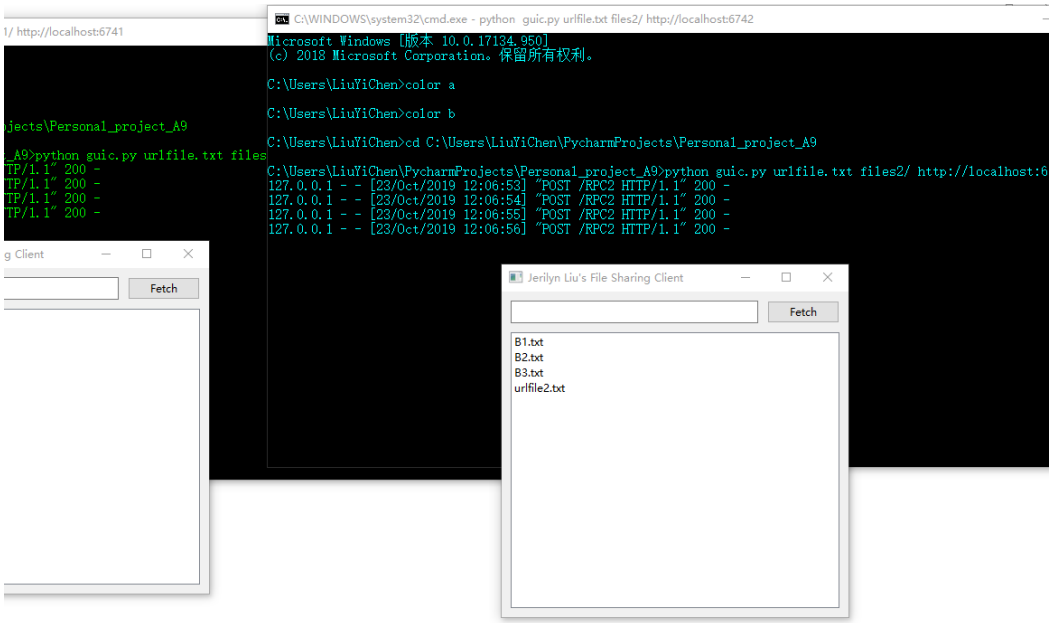


我们可以看到，GUI 界面已经形成，窗口的名称为自己设定的“Jerilyn Liu’s File Sharing Client”,同时，空白框可用于输入你想要获取的文件的文件名，下方的四个文件是 files1 目录下已有的文件。至此，我们已经开启了一个节点，接着再开启剩余两个节点。

2. 启动节点 http://localhost:6742

-Cd C:\Users\LiuYiChen\PycharmProjects\Personal_project_A9

- python guic.py urlfile.txt files2/ http://localhost:6742 #启动第二个节点

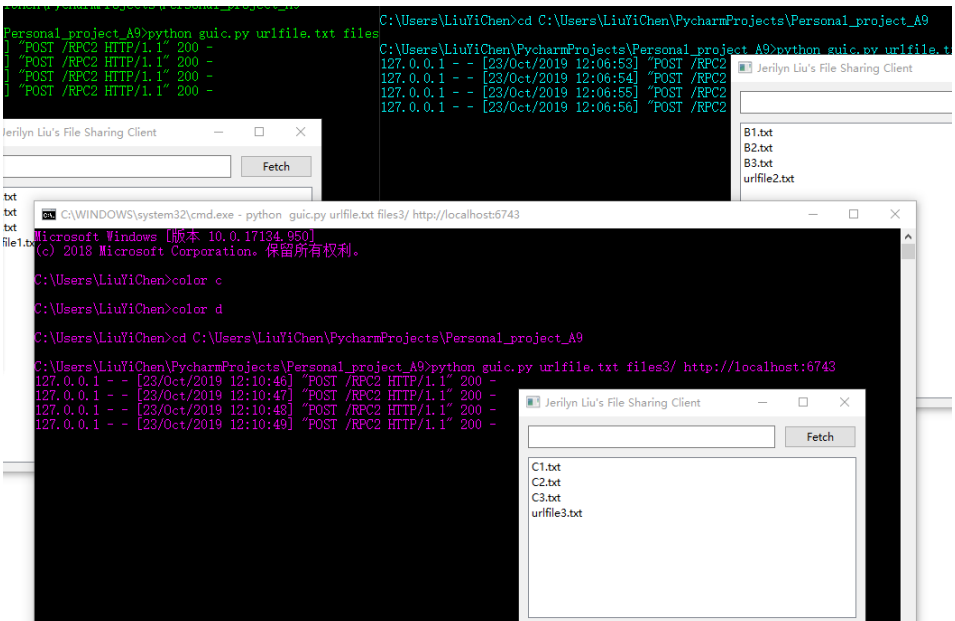


如图，第二个节点已启动完毕，同时 GUI 界面里显示了 files2 文件夹下可用的四个 txt 文件。

3. 启动节点 http://localhost:6743

-Cd C:\Users\LiuYiChen\PycharmProjects\Personal_project_A9

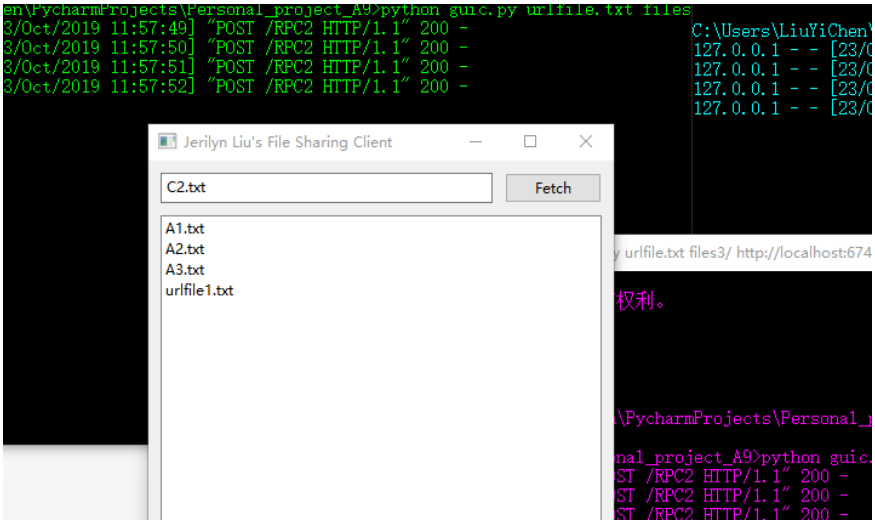
- python guic.py urlfile.txt files3/ http://localhost:6743 #启动第三个节点



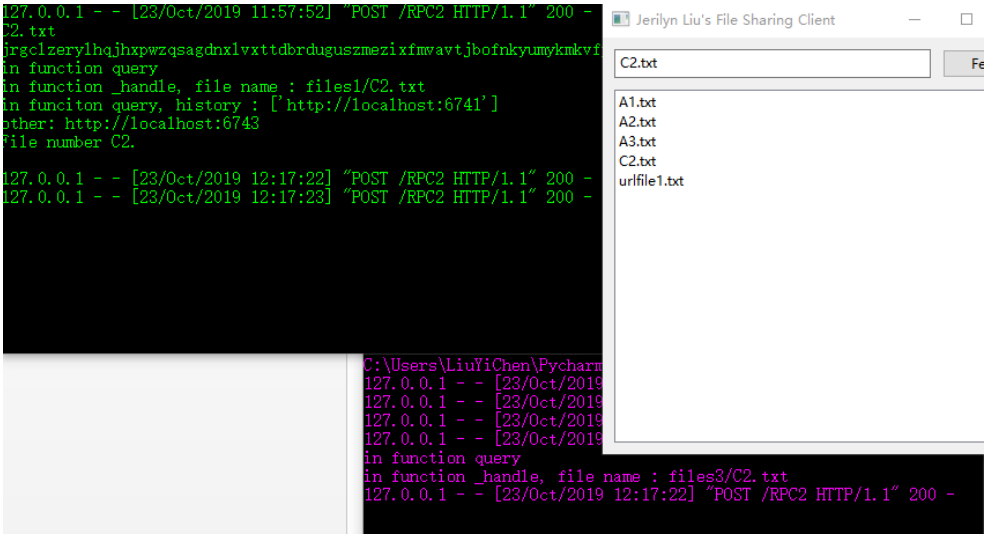
至此，三个节点已启动完毕，接下来可以使用 GUI 客户端进行文件传输了。

首先，我们测试将 files3 文件夹里的 C2.txt 文件下载到 files1 里。

在节点 6741 的 GUI 界面上输入 C2.txt，点击 Fetch 按钮：

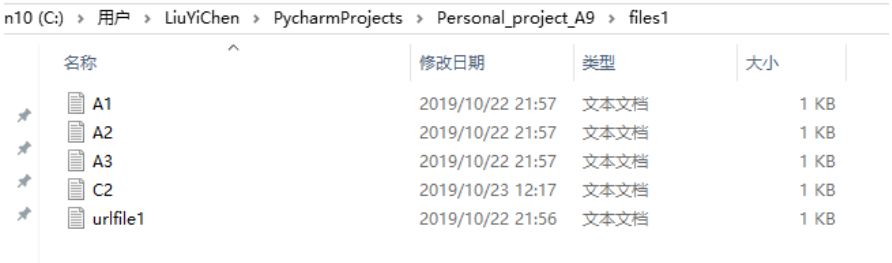


接着我们会发现，现在 files1 文件夹目录里多出了 C2.txt 这个可用文件。



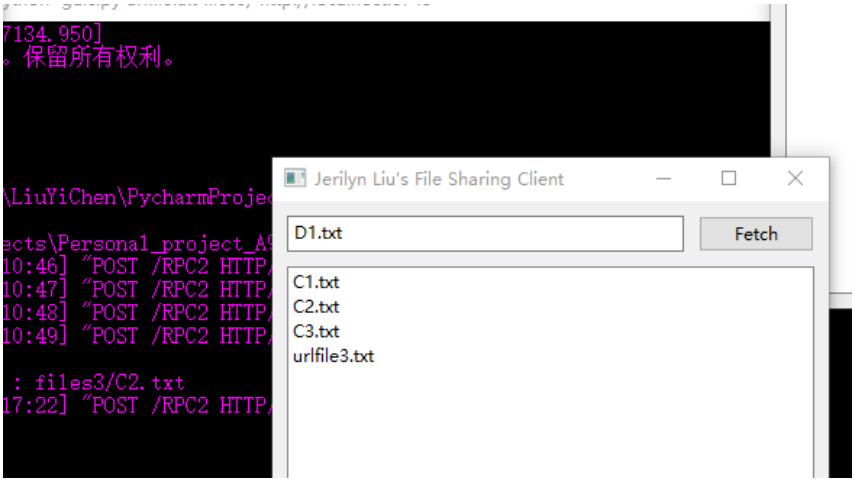
我们回到主机上，进入 files1 文件夹检查，得到了同样的结果。

C2.txt 已经被下载到 files1 里面了。如下所示：

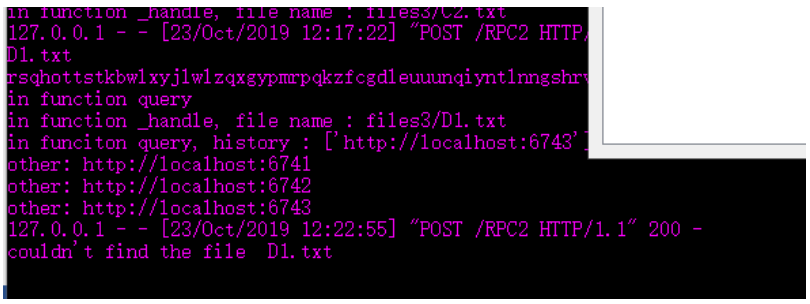


至此，我们已经完成了文件传输的测试。

为了测试程序的其他功能，我们尝试输入一个错误的文件名，即三个节点都不拥有的文件。比如：D1.txt



点击 Fetch 后，我们得到报错结果：



终端报错： “couldn't find the file D1.txt”

五．结果分析

在运行结果部分，我们测试了本机上的由三个节点组成的虚拟网络之间的文件传输。我们以节点 6741 和节点 6743 之间的连接为例子，实现了 C2.txt 的文件共享。同时，我们通过请求获取一个不存在的文件，测试了程序的报错功能。因此，从运行结果来看，我们已然实现了利用 XML-RPC 进行远程文件调用，以及为此共享程序添加了 GUI 的目的。

六．小结

在本程序中，我们利用 Pycharm+Anaconda 的环境，通过利用 XML-RPC 技术，加上 GUI 支持，搭建了一个远程文件共享程序。在整个过程中，需求分析阶段是关键的，我们详细分析了现实生活中文件传输的情景，阐释了用户的实际需要，进行了人性化的设置。另外，我们通过框图设计，表示了实现本程序所要利用的三个系统，以及三者之间的关系，比如客户端和服务端的文件传输协议。完整的代码 py 文件将附在本文后。

七. 参考文献

[1] :https://www.jianshu.com/p/2fc47e4a2bbb

[2]: https://www.jianshu.com/p/c2a8d293252e

[3]:https://baike.baidu.com/item/%E7%82%B9%E5%AF%B9%E7%82%B9%E6%8A%80%E6%9C%AF/8233664?fromtitle=p2p%E6%8A%80%E6%9C%AF&fromid=406788