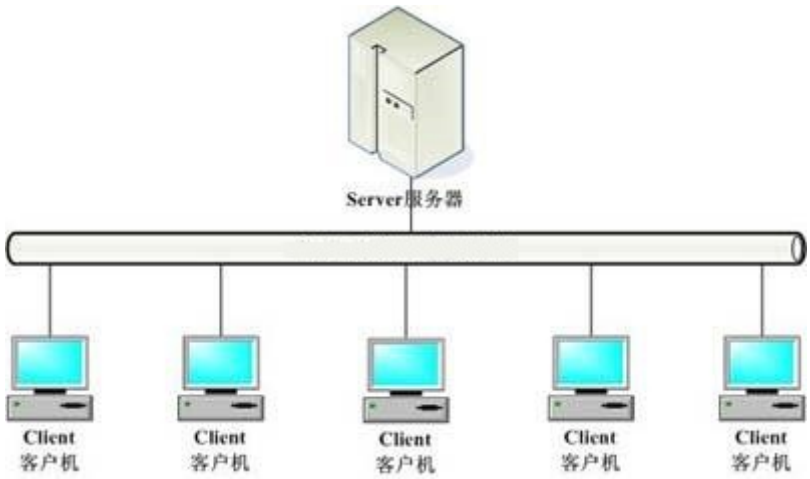
4. 网络并发模型

4.1 网络并发模型概述

- 什么是网络并发

  在实际工作中，一个服务端程序往往要应对多个客户端同时发起访问的情况。如果让服务端程序能够更好的同时满足更多客户端网络请求的情形，这就是并发网络模型。



- 循环网络模型问题

  循环网络模型只能循环接收客户端请求，处理请求。同一时刻只能处理一个请求，处理完毕后再处理下一个。这样的网络模型虽然简单，资源占用不多，但是无法同时处理多个客户端请求，是其最大的弊端，往往只有在一些低频的小请求任务中才会使用。

4.2 多进程/线程并发模型

多进程/线程并发模中每当一个客户端连接服务器，就创建一个新的进程/线程为该客户端服务，客户端退出时再销毁该进程/线程，多任务并发模型也是实际工作中最为常用的服务端处理模型。

- 模型特点

    - 优点：能同时满足多个客户端长期占有服务端的需求，可以处理各种请求
    - 缺点： 资源消耗较大
    - 适用情况：客户端请求较复杂，需要长时间占有服务器
- 创建流程
    - 创建网络套接字
    - 等待客户端连接
    - 有客户端连接，则创建新的进程/线程具体处理客户端请求
    - 主进程/线程继续等待处理其他客户端连接
    - 如果客户端退出，则销毁对应的进程/线程

多进程并发模型示例：

```
"""
基于多进程的网络并发模型
重点代码 ！！

创建tcp套接字
等待客户端连接
有客户端连接，则创建新的进程具体处理客户端请求
父进程继续等待处理其他客户端连接
如果客户端退出，则销毁对应的进程
"""
from socket import *
from multiprocessing import Process
import sys

# 地址变量
HOST = "0.0.0.0"
PORT = 8888
ADDR = (HOST, PORT)

# 处理客户端具体请求
def handle(connfd):
    while True:
        data = connfd.recv(1024)
        if not data:
            break
        print(data.decode())
    connfd.close()
```

```
# 服务入口函数
def main():
    # 创建tcp套接字
    tcp_socket = socket()
    tcp_socket.bind(ADDR)
    tcp_socket.listen(5)
    print("Listen the port %d"%PORT)

    # 循环连接客户端
    while True:
        try:
            connfd, addr = tcp_socket.accept()
            print("Connect from", addr)
        except KeyboardInterrupt:
            tcp_socket.close()
            sys.exit("服务结束")

        # 创建进程 处理客户端请求
        p = Process(target=handle, args=(connfd,),daemon=True)
        p.start()

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

多线程并发模型示例：

```
"""
基于多线程的网络并发模型
重点代码 ！！

思路： 网络构建     线程搭建     /    具体处理请求
"""
from socket import *
from threading import Thread


# 处理客户端具体请求
class Handle:
    # 具体处理请求函数 （逻辑处理，数据处理）
    def request(self, data):
        print(data)
```

```python
# 创建线程得到请求
class ThreadServer(Thread):
    def __init__(self, connfd):
        self.connfd = connfd
        self.handle = Handle()
        super().__init__(daemon=True)

    # 接收客户端的请求
    def run(self):
        while True:
            data = self.connfd.recv(1024).decode()
            if not data:
                break
            self.handle.request(data)
        self.connfd.close()


# 网络搭建
class ConcurrentServer:
    """
    提供网络功能
    """
    def __init__(self, *, host="", port=0):
        self.host = host
        self.port = port
        self.address = (host, port)
        self.sock = self.__create_socket()

    def __create_socket(self):
        tcp_socket = socket()
        tcp_socket.bind(self.address)
        return tcp_socket

    # 启动服务 --> 准备连接客户端
    def serve_forever(self):
        self.sock.listen(5)
        print("Listen the port %d" % self.port)

        while True:
```

```
            connfd, addr = self.sock.accept()
            print("Connect from", addr)
            # 创建线程
            t = ThreadServer(connfd)
            t.start()


if __name__ == '__main__':
    server = ConcurrentServer(host="0.0.0.0", port=8888)
    server.serve_forever()  # 启动服务
```

ftp 文件服务器

【1】 分为服务端和客户端，要求可以有多个客户端同时操作

【2】 客户端可以查看服务器文件库中有什么文件

【3】 客户端可以从文件库中下载文件到本地

【4】 客户端可以上传一个本地文件到文件库

【5】 使用print在客户端打印命令输入提示，引导操作

参考代码：

```
######################### 服务端 ##############################
from socket import *
from threading import Thread
import os
from time import sleep

# 文件库
FTP = "./FTP/"


# 处理客户端具体请求
class Handle:
    def __init__(self, connfd):
        self.connfd = connfd
```

```
    def do_list(self):
        filelist = os.listdir(FTP)
        if filelist:
            self.connfd.send(b"OK")
            sleep(0.1)
            # 发送文件列表
            files = "\n".join(filelist)
            self.connfd.send(files.encode())
        else:
            self.connfd.send(b"FAIL")

    def do_get(self, filename):
        try:
            file = open(FTP + filename, 'rb')
        except:
            self.connfd.send(b"FAIL")
        else:
            self.connfd.send(b"OK")
            sleep(0.1)
            #   发送文件
            while True:
                data = file.read(1024)
                if not data:
                    break
                self.connfd.send(data)
            file.close()
            sleep(0.1)
            self.connfd.send(b"##")

    def do_put(self, filename):
        # 判断文件是否存在
        if os.path.exists(FTP + filename):
            self.connfd.send(b"FAIL")
        else:
            self.connfd.send(b"OK")
            # 接收文件
            file = open(FTP + filename, 'wb')
            while True:
                data = self.connfd.recv(1024)
                if data == b"##":
                    break
```

```
                file.write(data)
            file.close()

    def request(self):
        while True:
            data = self.connfd.recv(1024).decode()
            # 分情况具体处理请求函数
            tmp = data.split(' ')
            if not data or tmp[0] == "EXIT":
                break
            elif tmp[0] == "LIST":
                self.do_list()
            elif tmp[0] == "GET":
                # tmp-> [GET,filename]
                self.do_get(tmp[1])
            elif tmp[0] == "PUT":
                self.do_put(tmp[1])


# 创建线程得到请求
class FTPThread(Thread):
    def __init__(self, connfd):
        self.connfd = connfd
        self.handle = Handle(connfd)
        super().__init__(daemon=True)

    # 接收客户端的请求
    def run(self):
        self.handle.request()
        self.connfd.close()


# 网络搭建
class ConcurrentServer:
    """
    提供网络功能
    """

    def __init__(self, *, host="", port=0):
        self.host = host
        self.port = port
```

```
        self.address = (host, port)
        self.sock = self.__create_socket()

    def __create_socket(self):
        tcp_socket = socket()
        tcp_socket.bind(self.address)
        return tcp_socket

    # 启动服务 --> 准备连接客户端
    def serve_forever(self):
        self.sock.listen(5)
        print("Listen the port %d" % self.port)

        while True:
            connfd, addr = self.sock.accept()
            print("Connect from", addr)
            # 创建线程
            t = FTPThread(connfd)
            t.start()


if __name__ == '__main__':
    server = ConcurrentServer(host="0.0.0.0", port=8880)
    server.serve_forever()  # 启动服务


############################ 客户端
##############################

"""
文件服务器客户端
"""
from socket import *
import sys
from time import sleep


# 具体发起请求，逻辑处理
class Handle:
    def __init__(self):
        self.server_address = ("127.0.0.1", 8880)
```

```
        self.sock = self.__connect_server()

    def __connect_server(self):
        tcp_socket = socket()
        tcp_socket.connect(self.server_address)
        return tcp_socket

    def do_list(self):
        self.sock.send(b"LIST")  # 发送请求
        response = self.sock.recv(1024)  # 接收响应
        if response == b"OK":
            # 接收文件列表 file1\nfile2\n..
            files = self.sock.recv(1024 * 1024)
            print(files.decode())
        else:
            print("获取文件列表失败")

    def do_exit(self):
        self.sock.send(b"EXIT")
        self.sock.close()
        sys.exit("谢谢使用")

    def do_get(self, filename):
        request = "GET " + filename
        self.sock.send(request.encode())  # 发送请求
        response = self.sock.recv(128)  # 接收响应
        if response == b"OK":
            file = open(filename, 'wb')
            # 接收文件内容，写入文件
            while True:
                data = self.sock.recv(1024)
                if data == b"##":
                    break
                file.write(data)
            file.close()
        else:
            print("该文件不存在")

    def do_put(self, filename):
        try:
            file = open(filename, 'rb')
```

```
        except:
            print("该文件不存在")
        else:
            filename = filename.split("/")[-1]  # 获取文件名
            request = "PUT " + filename
            self.sock.send(request.encode())
            response = self.sock.recv(128)
            if response == b"OK":
                # 发送文件
                while True:
                    data = file.read(1024)
                    if not data:
                        break
                    self.sock.send(data)
                file.close()
                sleep(0.1)
                self.sock.send(b"##")
            else:
                print("上传失败")


# 图形交互类
class FTPView:
    def __init__(self):
        self.__handle = Handle()

    def __display_menu(self):
        print()
        print("1. 查看文件")
        print("2. 下载文件")
        print("3. 上传文件")
        print("4. 退    出")
        print()

    def __select_menu(self):
        item = input("请输入选项:")
        if item == "1":
            self.__handle.do_list()
        elif item == "2":
            filename = input("要下载的文件:")
            self.__handle.do_get(filename)
```

```
            elif item == "3":
                filename = input("要上传的文件:")
                self.__handle.do_put(filename)
            elif item == "4":
                self.__handle.do_exit()
            else:
                print("请输入正确选项! ")

        def main(self):
            while True:
                self.__display_menu()
                self.__select_menu()


if __name__ == '__main__':
    ftp = FTPView()
    ftp.main()  # 启动
```

4.3 高并发技术探讨

- 衡量高并发的关键指标
  - 响应时间(Response Time)： 接收请求后处理的时间
  - 同时在线用户数量：同时连接服务器的用户的数量
  - 每秒查询率QPS(Query Per Second)： 每秒接收请求的次数
  - 每秒事务处理量TPS(Transaction Per Second)：每秒处理请求的次数（包含接收、处理、响应）
  - 吞吐量(Throughput)： 响应时间+QPS+同时在线用户数量