IO网络编程

==========================

| Tedu Python 教学部 |

| --- |

| Author：吕泽|

-----------

\*\*\*面试要求\*\*\*

\* OSI七层模型介绍一下，tcp/ip模型是什么？



\* tcp服务和udp服务有什么区别？

1、连接方面区别

TCP面向连接（如打电话要先拨号建立连接）。

UDP是无连接的，即发送数据之前不需要建立连接。

2、安全方面的区别

TCP提供可靠的服务，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达。

UDP尽最大努力交付，不保证可靠交付。

3、传输效率的区别

TCP传输效率相对较低。网页获取，文件下载，邮件收发。

UDP传输效率高，适用于对高速传输和实时性有较高的通信或广播通信。视频 直播 广播

4、连接对象数量的区别

TCP连接只能是点到点、一对一的。

UDP支持一对一，一对多，多对一和多对多的交互通信。

\* 三次握手和四次挥手指什么，过程是怎样的？

当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四次挥手。

1. 操作系统

\* 让应用层程序更好的访问硬件

\* 应用层 -> 操作系统 -> 硬件执行

\* Linux (开源) 命令操作

2. IO : IO密集 (cpu占用少,时间长,效率不高,可能有阻塞)

计算密集 (cpu占用大,运算速度块,几乎没有阻塞)

3. 文件的操作 (打开,读写,关闭)

打开 open() r w a + b

\* 字节串(bytes) encode() decode()

读写 read() readline() readlines()

write() writelines()

关闭 close() with语句

4. 缓冲区

减少磁盘交互次数,提高读写效率

flush() 刷新缓存

作业 : 1. 理论问题整理,会回答

2. 套接字一系列函数使用

3. 客户端可以循环的给服务端发送消息

[TOC]

## Linux 操作系统及其组成 **优点：**操作简介 安全性高

1. 操作系统的作用

操作系统（OS）是管理计算机硬件与软件资源的计算机程序，同时也是计算机系统的内核与基石。操作系统需要处理如管理与配置内存、决定系统资源供需的优先次序、控制输入设备与输出设备、操作网络与管理文件系统等基本事务。操作系统也提供一个让用户与系统交互的操作界面。

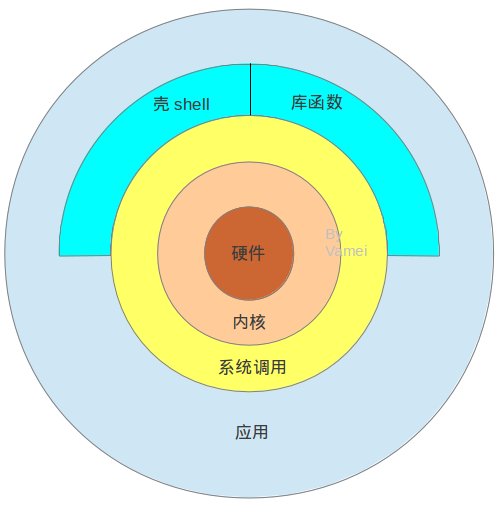
2. Linux操作系统组成

一个典型的Linux操作系统组成为：Linux内核，文件系统，命令行shell，图形界面和桌面环境，并包各种工具和应用软件。

\* Linux内核: Linux操作系统的核心代码 :

\* 文件系统：通常指称管理磁盘数据的系统，可将数据以目录或文件的型式存储。每个文件系统都有自己的特殊格式与功能

\* shell命令： 接收用户命令，然后调用相应的应用程序，并根据用户输入的指令来反馈给用户指定的信息。



## shell命令

### 文件操作命令

\* linux下的目录结构

![Linux](img/linux\_fs.jpg)

| 作用 | 命令 |

| --- | --- |

**| 切换工作目录 | cd | 绝对路径/ 相对路径**

**| 查看文件 | ls ， ls -l ， ls -a |**

| 复制文件 | cp -r |

| 移动文件 | mv |

| 删除文件 | rm -rf ， rmdir |

| 创建文件夹| mkdir -p |

| 创建文件| touch |

**| 查看文件内容| cat |**

## IO I/O输入/输出(Input/Output)，分为IO设备和IO接口两个部分。

1. 定义

>在内存中存在数据交换的操作认为是IO操作,比如和终端交互（input(),print()） ,和磁盘交互，和网络交互等

2. 程序分类

>\* IO密集型程序：在程序执行中有大量IO操作，而cpu运算较少。**消耗cpu较少，耗时长,效率低**。

>\* 计算密集型程序：程序运行中计算较多，IO操作相对较少。**cpu消耗多，执行速度快，几乎没有阻塞。（阻塞函数：input() sleep()）**

## 文件 (一切皆文件)

文件是保存在持久化存储设备(硬盘、U盘、光盘..)上的一段数据。从格式编码角度分为文本文件（打开后会自动解码为字符word txt .py）、二进制文件(视频、音频、图片等)。在Python里把文件视作一种类型的对象，类似之前学习过的其它类型。

并不是所有二进制都能转为文本文件

### 字节串（bytes）

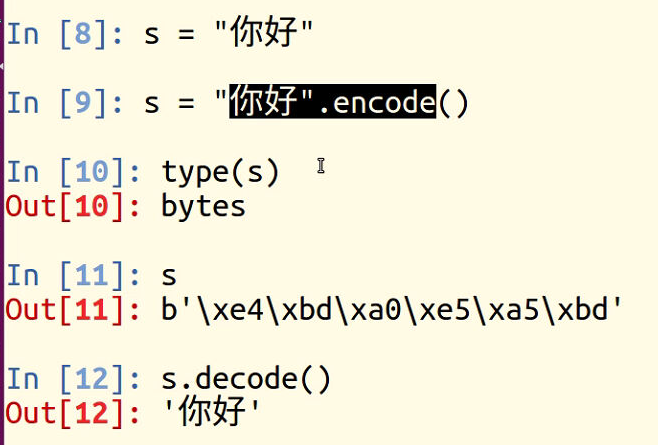
在python3中引入了字节串的概念，与str不同，字节串以字节序列值表达数据，更方便用来处理二进程数据。因此在python3中字节串是常见的**二进制数据展现方式。**

**一个汉字在python中占3个字节**

\* 普通的ascii编码字符串可以**在前面加b转换为字节串**，例如：b'hello'

\* 字符串转换为字节串方法 ：str.encode()

\* 字节串转换为字符串方法 : bytes.decode()



### 文件读写

对文件实现读写的基本操作步骤为：打开文件，读写文件，关闭文件

\*\*\*代码实现： day4/file\_open.py\*\*\*

*"""*

*file\_open.py*

*文件打开方式训练*

*"""*

*# 打开文件*

**try**:

**"""**

**文本文件既可以使用文本方式打开,也能使用二进进制方式打开**

**二进制文件如果使用文本方式打开,读写时会报错**

**"""**

*# f = open('text.py','r') # 只读方式*

*# f = open('text.py','w') # 只写方式*

f = open(**'text.py'**,**'a'**) *# 追加方式*

**except** Exception **as** e:

print(e)

*# 读写文件*

*# 关闭文件*

f.close()

1. 打开文件

```python

file\_object = open(file\_name, access\_mode='r', buffering=-1)

功能：打开一个文件，返回一个文件对象。

参数：file\_name 文件名；

access\_mode 打开文件的方式,如果**不写默认为‘r’**

**'''**

**文本文件既可以用文本方式打开，也可以使用二进制方式打开**

**二进制文件如果用文本方式打开，读写时会报错**

**'''**

文件模式 操作

**文本方式打开的是：字符串**

r 以读方式打开 文件必须存在 不存在则报错

w 以写方式打开 文件不存在则创建，存在清空原有内容

a 以追加模式打开

r+ 以读写模式打开 文件必须存在

w+ 以读写模式打开文件 不存在则创建，存在清空原有内容

a+ 以读写模式打开 追加模式

**二进制方式打开的是：字节串**

rb 以二进制读模式打开 同r

wb 以二进制写模式打开 同w

ab 以二进制追加模式打开 同a

rb+ 以二进制读写模式打开 同r+

wb+ 以二进制读写模式打开 同w+

ab+ 以二进制读写模式打开 同a+

buffering 1表示有行缓冲，默认则表示使用系统默认提供的缓冲机制。

返回值：成功返回文件操作对象。

1. 读取文件

>read([size])

>功能： 来直接读取文件中字符。

>参数： 如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取直至末尾，给定size最多读取给定数目个字符（字节）。

>返回值： 返回读取到的内容

>

>\* 注意：文件过大时候不建议直接读取到文件结尾，读到文件结尾会返回空字符串。

>readline([size])

>功能： 用来读取文件中一行

>参数： 如果没有给定size参数（默认值为-1）或者size值为负，表示读取一行，给定size表示最多读取制定的字符（字节）。

>返回值： 返回读取到的内容

>readlines([sizeint])

>功能： 读取文件中的每一行作为列表中的一项

>参数： 如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取直至末尾，给定size表示读取到size字符所在行为止。

>返回值： 返回读取到的内容列表

>文件对象本身也是一个可迭代对象，在for循环中可以迭代文件的每一行。

```python

for line in f:

print(line)

```

\*\*\*代码实现： day4/file\_read.py\*\*\*

*"""*

*file\_read.py*

*文件读取演示*

*"""*

*# 打开文件*

f = open(**'1.jpg'**,**'rb'**)

*# 读操作*

**while True**:

*# 到文件结尾时会读出空字串*

data = f.read(16)

*# 到文件结尾跳出循环*

**if not** data:

**break**

print(**"读取到的数据:"**,data)

*# 每次读取一行内容*

*# data = f.readline(6)*

*# print("一行内容:",data)*

*# data = f.readline()*

*# print("一行内容:",data)*

*# 将内容读取到一个列表*

*# 参数表达的是读取到该字符数所在的行*

*# data = f.readlines(28)*

*# print(data)*

*# 文件对象可迭代,每次一行*

*# for line in f:*

*# print(line)*

*# 关闭*

f.close()

**练习:在字典中查找单词对应行**

*"""*

*从终端输入一个单词,从单词本中刚找到该单词,打印这一行内容,如果没有找到则打印"找不到"*

*"""*

word = input(**"单词:"**)

*# 打开文件*

f = open(**'dict.txt'**)

**for** line **in** f:

w = line.split(**' '**)[0]

*# 遍历的单词已经大于目标,说明找不到了*

**if** w > word:

print(**"没有找到该单词"**)

**break**

**elif** w == word:

print(line)

**break**

**else**:

print(**"没有找到该单词"**)

f.close()

3. 写入文件

>write(string)

>功能: 把文本数据或二进制数据块的字符串写入到文件中去

>参数：要写入的内容

>返回值：写入的字符个数

>

>\* 如果需要换行要自己在写入内容中添加\n

>writelines(str\_list)

>功能：接受一个字符串列表作为参数，将它们写入文件。

>参数: 要写入的内容列表

\*\*\*代码实现： day4/file\_write.py\*\*\*

*"""*

*file\_write.py*

*文件写操作演示*

*"""*

f = open(**'text'**,**'w'**)

*# f = open('text','ab')*

*# 写操作*

*# f.write(b"hello,diegui\n") # 如果希望换行则自己添加*

*# f.write("哎呀,干啥".encode())*

*# 写入列表内容*

l = [**'hello world\n'**,**'哈哈哈\n'**] *#将代码输出到一行,如果想换行需自己添加\n*

f.writelines(l)

f.close()

4. 关闭文件

打开一个文件后我们就可以通过文件对象对文件进行操作了，当操作结束后使用close（）关闭这个对象可以防止一些误操作，也可以节省资源。

>file\_object.close()

**练习:拷贝文件**

*"""*

*编写一个文件拷贝程序,将一个文件拷贝一份,重新取另外一个名字(自定).*

*文件可能是文本,也可能是二进制*

*"""*

filename = input(**"文件:"**)

fr = open(filename,**'rb'**)

fw = open(**"备份-"**+filename,**'wb'**)

**while True**:

*# 循环读取*

data = fr.read(1024)

**if not** data: *# 文件结束*

**break**

fw.write(data)

fr.close()

fw.close()

5. with操作

python中的with语句使用于对资源进行访问的场合，保证不管处理过程中是否发生错误或者异常都会执行规定的“清理”操作，释放被访问的资源，比如有文件读写后自动关闭、线程中锁的自动获取和释放等。

with语句的语法格式如下：

```python

with context\_expression [as obj]:

with-body

```

通过with方法可以不用close(),因为with生成的对象在语句块结束后会自动处理，所以也就不需要close了，但是这个文件对象只能在with语句块内使用。

```python

with open('file','r+') as f:

f.read()

```

*"""*

*with.py*

*with语句生成文件对象*

*"""*

**with** open(**'4.txt'**) **as** f: *# 以只读方式生成f对象*

data = f.read()

print(data)

*# with语句块结束,f会被自动清理*

>注意

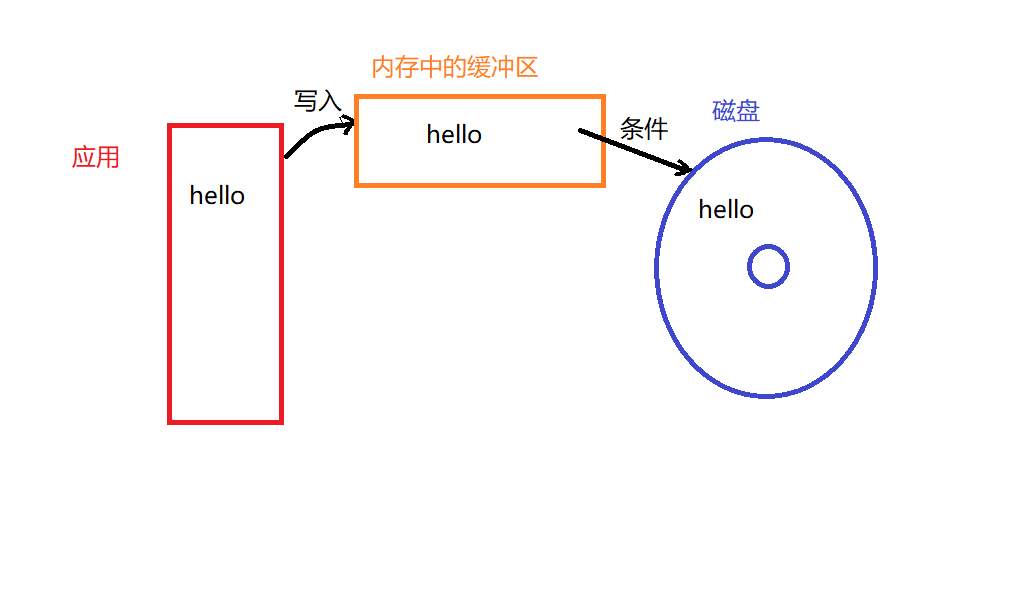
>> 1. 加b的打开方式读写要求必须都是字节串

>> 2. 无论什么文件都可以使用二进制方式打开，但是二进制文件使用文本方式打开读写会出错

### 其他操作

#### 刷新**缓冲区 作用:减少磁盘交互次数 提高读写效率**

缓冲:系统自动的在内存中为每一个正在使用的文件开辟一个缓冲区，从内存向磁盘输出数据必须先送到内存缓冲区，再由缓冲区送到磁盘中去。从磁盘中读数据，则一次从磁盘文件将一批数据读入到内存缓冲区中，然后再从缓冲区将数据送到程序的数据区。



刷新缓冲区条件：

1. 缓冲区被写满

2. 程序执行结束或者文件对象被关闭

3. 行缓冲遇到换行

4. 程序中调用flush()函数

\*\*\*代码实现： day4/buffer.py\*\*\*

*"""*

*buffer.py*

*缓冲区演示*

*"""*

*# f = open('test','w',1) # 按一次换行缓存一次*

f = open(**'test'**,**'w'**)

**while True**:

data = input(**">>"**)

**if not** data:

**break**

f.write(data + **'\n'**)

f.flush() *# 主动刷新缓存*

f.close()

>flush()

该函数调用后会进行一次磁盘交互，将缓冲区中的内容写入到磁盘。

#### 文件偏移量

\*\*\*代码实现： day4/seek.py\*\*\*

*'''*

*seek.py 文件偏移量*

*注意:1.open打开文件会重置文件偏移量*

*2.读写操作使用的是一个偏移量值*

*3.seek操作文件偏移量一般是以二进制打开*

*'''*

f=open(**'test'**, **'wb+'**)

f.write(**b'hello world'**)

print(“偏移量:”,f.tell()) *# 获取文件偏移量*

*# f.seek(0, 0) # 将文件偏移量定位到开头*

*# f.seek(5, 0) # 将文件偏移量定位到第五个*

print(f.seek(-1, 2)) *#必须以二进制方式打开文件时基准位置才能是1或者2*

data=f.read()

print(data)

f.close()

1. 定义

>打开一个文件进行操作时系统会自动生成一个记录，记录中描述了我们对文件的一系列操作。其中包括每次操作到的文件位置。文件的读写操作都是从这个位置开始进行的。

2. 基本操作

>tell()

功能：获取文件偏移量大小

>seek(offset[,whence])

>功能:移动文件偏移量位置

>参数：offset 代表相对于某个位置移动的字节数。负数表示向前移动，正数表示向后移动。

> whence是基准位置的默认值为 0，代表从文件开头算起，1代表从当前位置算起，2 代表从文件末尾算起。

>

```python

with open('file','r+') as f:

f.read()

```

>\* 必须以二进制方式打开文件时基准位置才能是1或者2

#### 文件描述符

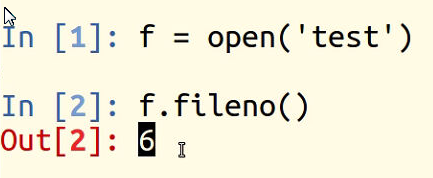
1. 定义

>系统中每一个IO操作都会分配一个整数作为编号，该整数即这个IO操作的文件描述符。

2. 获取文件描述符

>fileno()

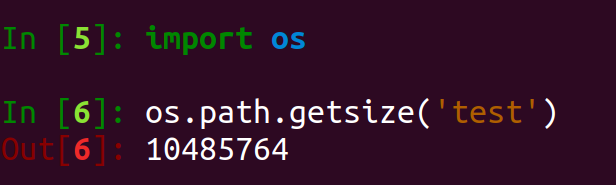
通过IO对象获取对应的文件描述符



### 文件管理函数

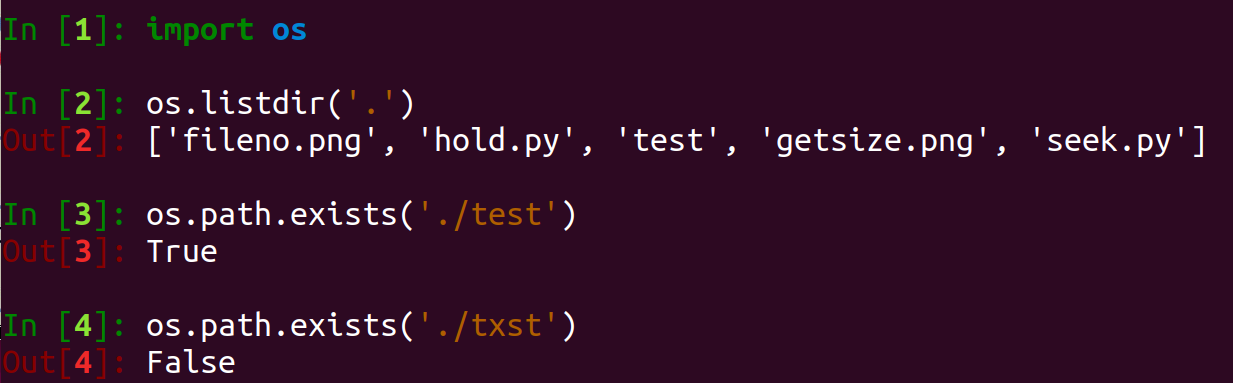
1. 获取文件大小

>os.path.getsize(file)



2. 查看文件列表

>os.listdir(dir)

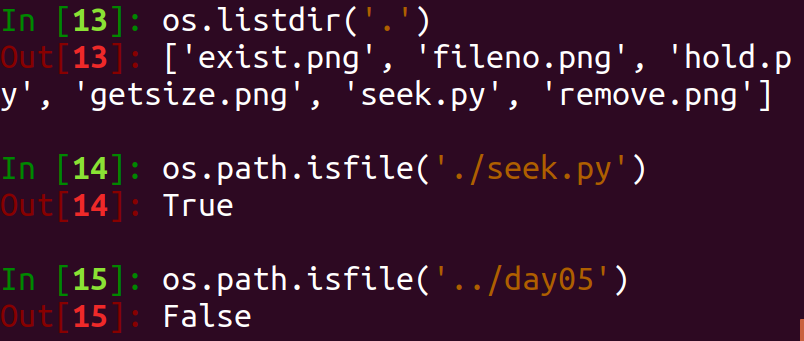


3. 查看文件是否存在

>os.path.exists(file)

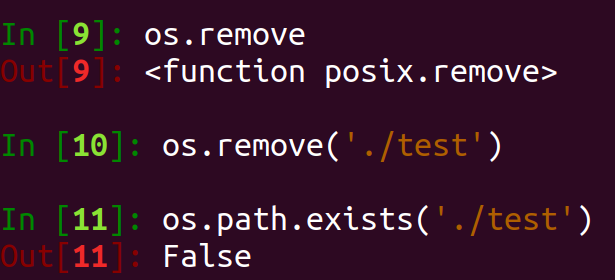
4. 判断文件类型

>os.path.isfile(file)



5. 删除文件

>os.remove(file)



*"""*

*运行程序时,写一个日志文件,格式如下*

*1. Fri Aug 30 17:57:45 2019*

*2. Fri Aug 30 17:57:46 2019*

*3. Fri Aug 30 17:57:47 2019*

*4. Fri Aug 30 17:57:48 2019*

*5. Fri Aug 30 17:57:58 2019*

*每隔一秒写依次,每个时间占一行*

*当程序终止运行,重写启动的时候,序列号能够衔接*

*"""*

**import** time

f = open(**'log.txt'**,**'a+'**)

f.seek(0) *# 将文件偏移量放到最开始*

n = len(f.readlines())

**while True**:

n += 1

time.sleep(1)

s = **"%d. %s\n"**%(n,time.ctime())

f.write(s)

f.flush() *# 刷新缓存*

## 网络编程基础

计算机网络功能主要包括实现资源共享，实现数据信息的快速传递。

### OSI七层模型

>制定组织： ISO（国际标准化组织）

>作用：使网络通信工作流程标准化

>应用层 ： 提供功能(qq/email/交友软件) 提供用户服务，具体功能由应用程序实现

>表示层 ： 数据的压缩/优化/加密

>会话层 ： 建立用户级的连接，选择适当的传输服务

>传输层 ： 提供传输服务

网络工程师的工作

>网络层 ： 路径查找 路由选择，网络互联

>链路层 ： 进行数据交换，控制具体数据的发送

>物理层 ： 提供数据传输的硬件保证，网卡接口，传输介质



>优点

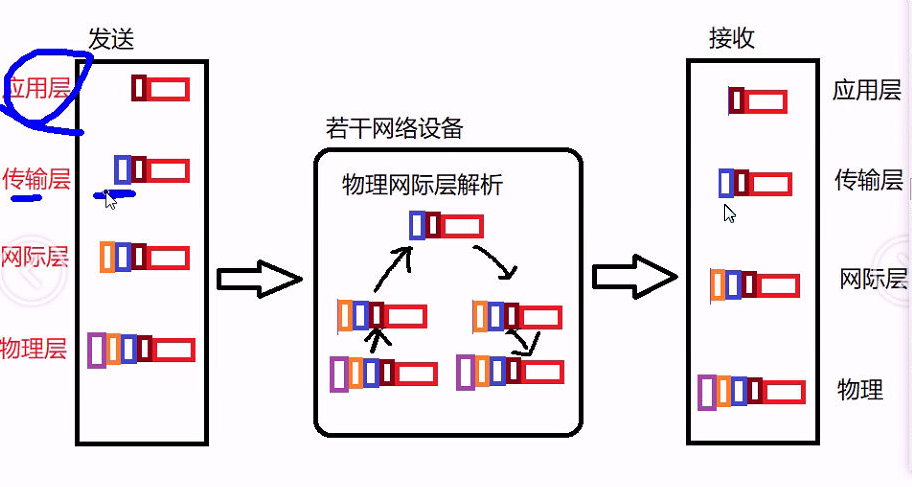
>1. 建立了统一的工作流程

>2. 分部清晰，各司其职，每个步骤分工明确

>3. 降低了各个模块之间的耦合度，便于开发

### 四层模型（TCP/IP模型）

背景 ： 实际工作中工程师无法完全按照七层模型要求操作，逐渐演化为更符合实际情况的四层

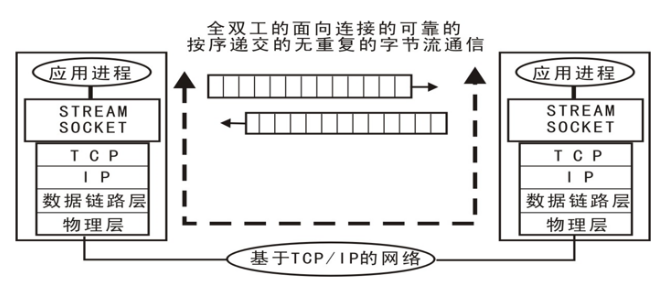


#### 数据传输过程

1. 发送端由应用程序发送消息，逐层添加首部信息，最终在物理层发送消息包。

2. 发送的消息经过多个节点（交换机，路由器）传输，最终到达目标主机。

3. 目标主机由物理层逐层解析首部消息包，最终到应用程序呈现消息。



#### 网络协议

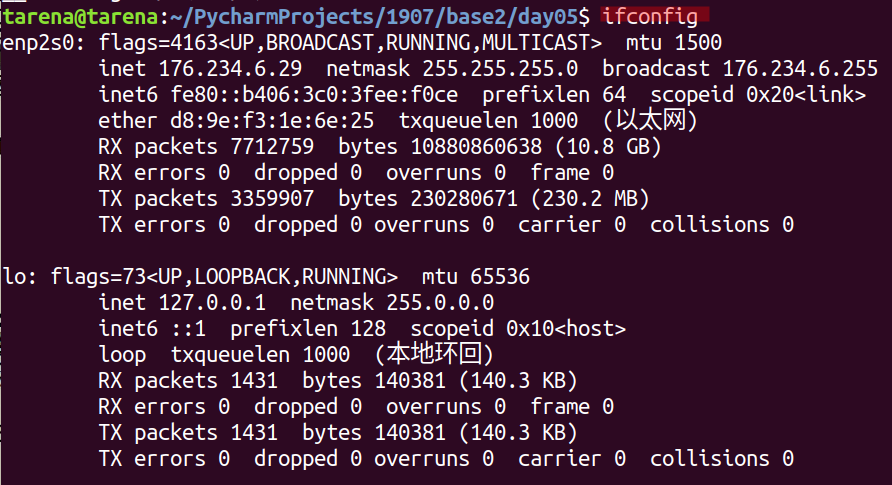
>在网络数据传输中，都遵循的规定，包括建立什么样的数据结构，什么样的特殊标志等。

### 网络基础概念

\* IP地址

>功能：确定一台主机的网络路由位置

>查看本机网络地址命令： ifconfig



>结构

>>IPv4 点分十进制表示 172.40.91.185 每部分取值范围0--255 ,(4\*8)=32位.2共有2\*\*32中可能. [一个2进制占8位,一个16进制占4位]

>>IPv6 128位 扩大了地址范围

inet 176.234.6.29

inet6 fe80::b406:3c0:3fee:f0ce

\* 域名 即 网址/url

>定义： 给网络服务器地址起的名字

>作用： 方便记忆，表达一定的含义

>ping [ip] : 测试和某个主机是否联通

\* 端口号（port）

>作用：端口是网络地址的一部分，用于区分主机上不同的网络应用程序。

>特点：一个系统中的应用监听端口不能重复

>取值范围： 1 -- 65535

>>1--1023 系统应用或者大众程序监听端口

>>1024--65535 自用端口

## 传输层服务

### 面向连接的传输服务（基于TCP协议的数据传输）

1. 传输特征 ： 提供了可靠的数据传输，可靠性指数据传输过程中无丢失，无失序，无差错，无重复。

2. 实现手段 ： 在通信前需要建立数据连接，通信结束要正常断开连接。

> 三次握手（建立连接）

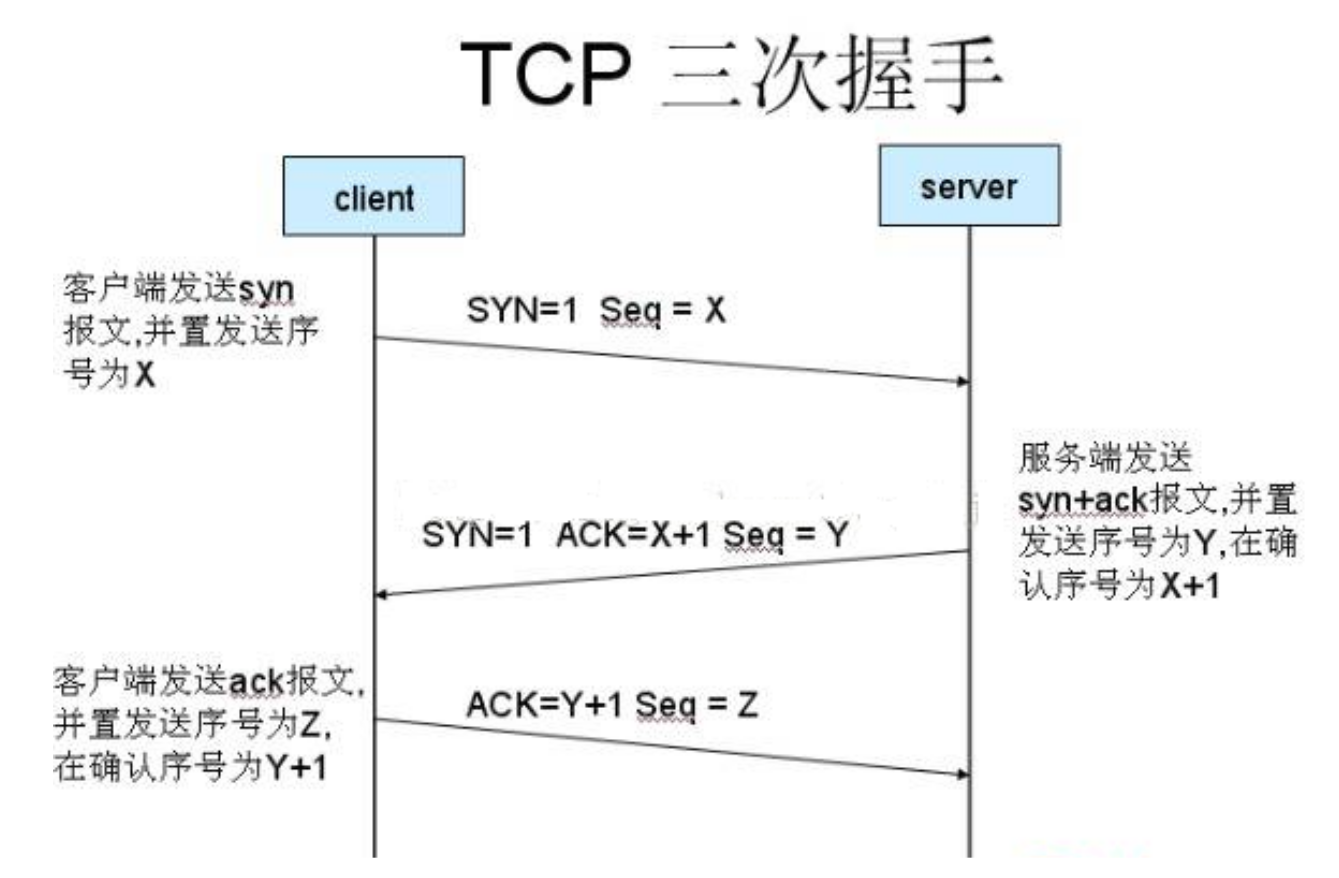
>>客户端向服务器发送消息报文请求连接

>>服务器收到请求后，回复报文确定可以连接

>>客户端收到回复，发送最终报文连接建立

客户端:client

服务端:server



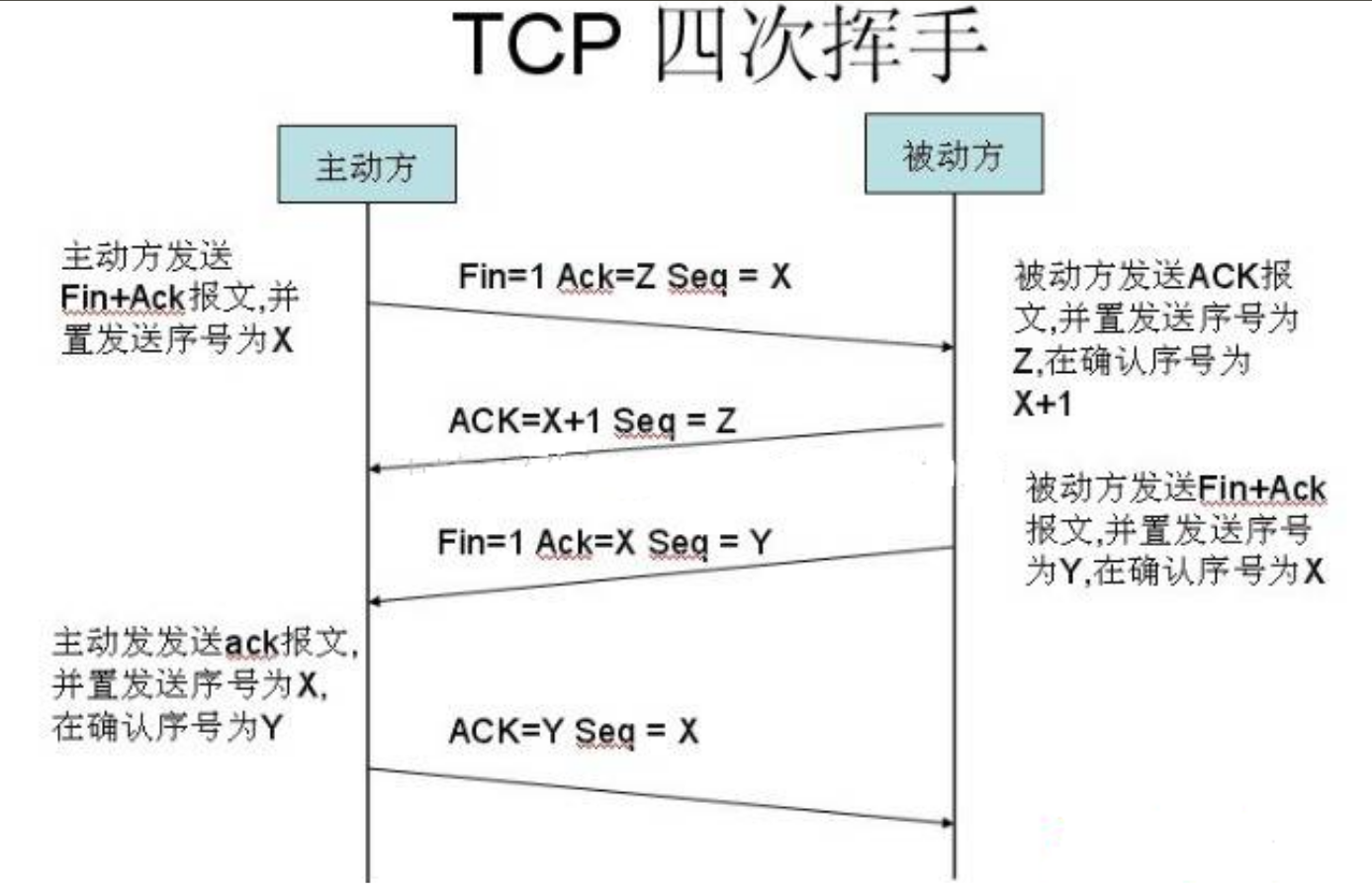
>四次挥手（断开连接）

>>主动方发送报文请求断开连接

>>被动方收到请求后，立即回复，表示准备断开

>>被动方准备就绪，再次发送报文表示可以断开

>>主动方收到确定，发送最终报文完成断开



3. 适用情况 ： 对数据传输准确性有明确要求，传数文件较大，需要确保可靠性的情况。比如：网页获取，文件下载，邮件收发。

### 面向无连接的传输服务（基于UDP协议的数据传输）

1. 传输特点 ： 不保证传输的可靠性，传输过程没有连接和断开，数据收发自由随意。

2. 适用情况 ： 网络较差(丢包,丢帧)，对传输可靠性要求不高。比如：网络视频，群聊，广播

## socket套接字编程

### 套接字介绍

1. 套接字 ： 实现网络编程进行数据传输的一种技术手段

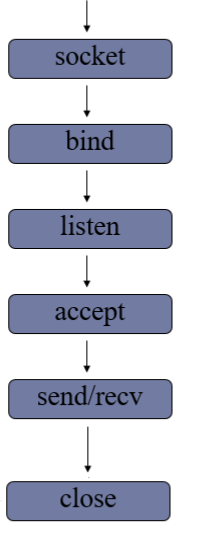
2. Python实现套接字编程：import socket

3. 套接字分类

>流式套接字(SOCK\_STREAM): 以字节流方式传输数据，实现tcp网络传输方案。(面向连接--tcp协议--可靠的--流式套接字)

>数据报套接字(SOCK\_DGRAM):以数据报形式传输数据，实现udp网络传输方案。(无连接--udp协议--不可靠--数据报套接字)

### tcp套接字编程

#### 服务端流程

\*\*\*代码实现：day5/tcp\_server.py\*\*\*

*"""*

*tcp\_server.py tcp套接字服务端流程*

*重点代码*

*注意: 功能性代码,注重流程和函数使用*

*"""*

**import** socket

*# 创建tcp套接字对象*

sockfd = socket.socket(socket.AF\_INET,

socket.SOCK\_STREAM)

*# 绑定地址*

sockfd.bind((**'0.0.0.0'**,9999))

*# 设置监听*

sockfd.listen(5)

*# 等待处理客户端连接请求*

**while True**:

print(**"Waiting for connect..."**)

**try**:

connfd,addr = sockfd.accept()

print(**"Connect from"**,addr)

**except** KeyboardInterrupt:

*# ctrl-c 退出程序*

print(**"Server exit"**)

**break**

**except** Exception **as** e:

print(e)

**continue**

*# 消息收发*

**while True**:

data = connfd.recv(1024)

*# 如果data为空意味着客户端断开*

**if not** data:

**break**

print(**"Receive:"**,data.decode())

*# if data == b'Q':*

*# break*

n = connfd.send(**b"Thanks"**)

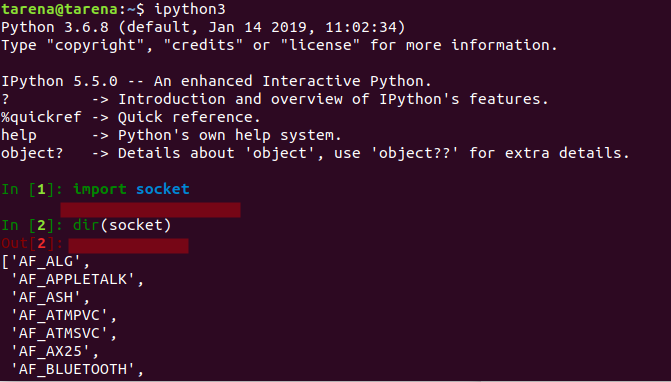
print(**'Send %d bytes'**%n)

connfd.close()

*# 关闭套接字*

sockfd.close()

1. 创建套接字



```python

sockfd=socket.socket(socket\_family=AF\_INET,socket\_type=SOCK\_STREAM,proto=0)

功能：创建套接字

参数： socket\_family 网络地址类型 AF\_INET表示ipv4

socket\_type 套接字类型 SOCK\_STREAM(流式) SOCK\_DGRAM(数据报)

proto 通常为0 选择子协议

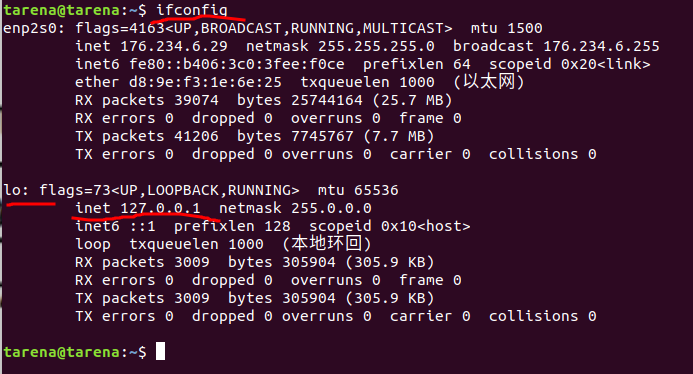
返回值： 套接字对象

```

2. 绑定地址

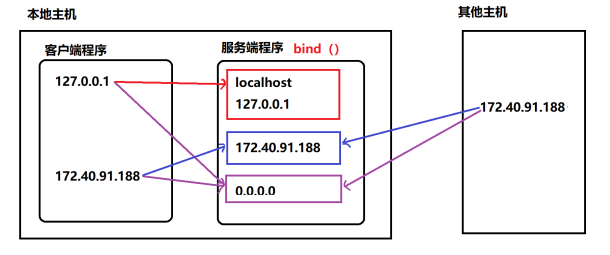
>本地地址 ： 'localhost' , '127.0.0.1'

本地测试地址查看方法:



>网络地址 ： '176.234.6.29'

>自动获取地址： '0.0.0.0'



```python

sockfd.bind(addr)

功能： 绑定本机网络地址

ip,端口号

参数： 二元元组 (ip,port) ('0.0.0.0',8888)

```

3. 设置监听

```python

sockfd.listen(n)

功能 ： 将套接字设置为监听套接字，确定监听队列大小

参数 ： 监听队列大小

```

4. 等待处理客户端连接请求

```python

connfd,addr = sockfd.accept()

功能： 阻塞等待处理客户端请求

返回值： connfd 客户端连接套接字

addr 连接的客户端地址

```

5. 消息收发

```python

data = connfd.recv(buffersize)

功能 : 接受客户端消息

参数 ：每次最多接收消息的大小

返回值： 接收到的内容

n = connfd.send(data)

功能 : 发送消息

参数 ：要发送的内容 bytes格式

返回值： 发送的字节数

```

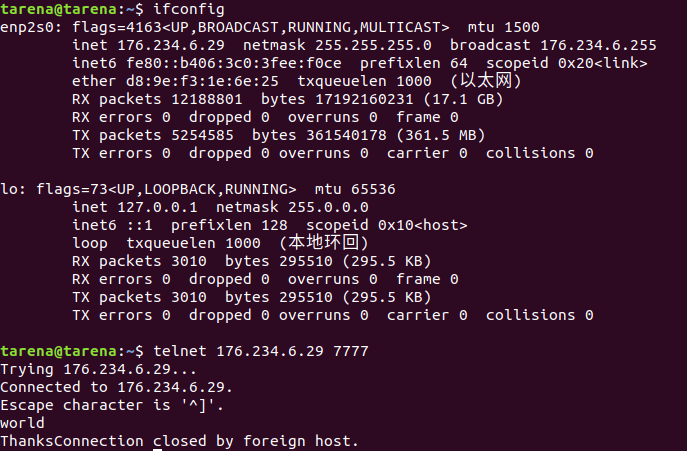
6. 关闭套接字

```python

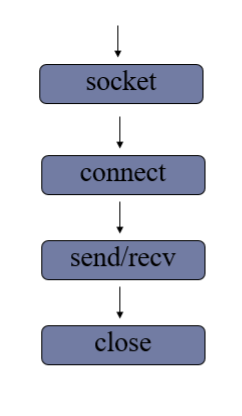
sockfd.close()

功能：关闭套接字

```



#### 客户端流程



\*\*\*代码实现：day5/tcp\_client.py\*\*\*

*"""*

*tcp\_client.py tcp套接字客户端流程*

*重点代码*

*注意: 和服务端配合,使用同样的套接字*

*"""*

**from** socket **import** \*

*# 创建tcp套接字*

sockfd = socket() *# 默认值*

*# 连接服务器*

server\_addr = (**'127.0.0.1'**,9999) *# 服务器地址*

sockfd.connect(server\_addr)

*# 先发后收*

**while True**:

msg = input(**"Msg:"**)

**if not** msg:

**break**

sockfd.send(msg.encode()) *#字节串*

*# if msg == 'Q':*

*# break*

data = sockfd.recv(1024)

print(**"From server:"**,data.decode())

sockfd.close()

1. 创建套接字

>注意:只有相同类型的套接字才能进行通信

2. 请求连接

```python

sockfd.connect(server\_addr)

功能：连接服务器

参数：元组 服务器地址

```

3. 收发消息

>注意： 防止两端都阻塞，recv send要配合

4. 关闭套接字

#### tcp 套接字数据传输特点

>\* tcp连接中当一端退出，另一端如果阻塞在recv，此时recv会立即返回一个空字串。

>\* tcp连接中如果一端已经不存在，仍然试图通过send发送则会产生BrokenPipeError

>\* 一个监听套接字可以同时连接多个客户端，也能够重复被连接

#### 网络收发缓冲区

1. 网络缓冲区有效的协调了消息的收发速度

2. send和recv实际是向缓冲区发送接收消息，当缓冲区不为空recv就不会阻塞。

#### tcp粘包

>原因：tcp以字节流方式传输，没有消息边界。多次发送的消息被一次接收，此时就会形成粘包。

>影响：如果每次发送内容是一个独立的含义，需要接收端独立解析此时粘包会有影响。

>处理方法

>>1. 人为的添加消息边界

>>2. 控制发送速度

*"""*

*练习* *: 将一个文件从客户端发送给服务端*

*"""*

*"""*

*发送一个文件*

*思路* *: 循环读取文件内容,发送*

*"""*

**from** socket **import** \*

s = socket()

s.connect((**'127.0.0.1'**,8888))

f = open(**'timg.jpeg'**,**'rb'**)

**while True**:

*# 边读 边发*

data = f.read(1024)

**if not** data: *# 文件结尾*

**break**

s.send(data)

f.close()

s.close()

*"""*

*接受一个文件*

*思路* *: 接收文件内容,将其写入到一个文件里*

*"""*

**from** socket **import** \*

s = socket()

s.bind((**'127.0.0.1'**,8888))

s.listen(3)

c,addr = s.accept()

print(**"Connect from"**,addr)

*# 打开文件*

f = open(**'mm.jpg'**,**'wb'**)

*# 循环接收内容,写入文件*

**while True**:

data = c.recv(1024)

**if not** data:

**break**

f.write(data)

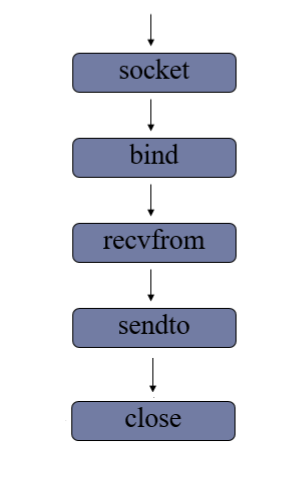
f.close()

c.close()

s.close()

### UDP套接字编程

#### 服务端流程



\*\*\*代码实现：day6/udp\_server.py\*\*\*

*"""*

*udp\_server.py udp套接字服务端流程*

*重点代码*

*"""*

**from** socket **import** \*

*# 创建udp套接字*

sockfd = socket(AF\_INET,SOCK\_DGRAM)

*# 绑定地址*

server\_addr = (**'127.0.0.1'**,8888)

sockfd.bind(server\_addr)

*# 循环收发消息*

**while True**:

data,addr = sockfd.recvfrom(1024)

print(**"Msg from %s: %s"**%(addr,data.decode()))

sockfd.sendto(**b'Thanks'**,addr)

*# 关闭套接字*

sockfd.close()

1. 创建数据报套接字

```python

sockfd = socket(AF\_INET,SOCK\_DGRAM)

```

2. 绑定地址

```python

sockfd.bind(addr)

```

3. 消息收发

```python

data,addr = sockfd.recvfrom(buffersize)

功能： 接收UDP消息

参数： 每次最多接收多少字节

返回值： data 接收到的内容

addr 消息发送方地址

n = sockfd.sendto(data,addr)

功能： 发送UDP消息

参数： data 发送的内容 bytes格式

addr 目标地址

返回值：发送的字节数

```

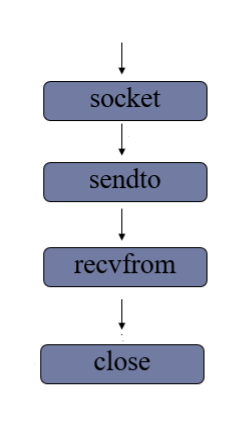
4. 关闭套接字

```python

sockfd.close()

```

#### 客户端流程



\*\*\*代码实现：day6/udp\_client.py42 5 14\*\*

*"""*

*udp\_client.py udp客户端*

*重点代码*

*"""*

**from** socket **import** \*

*# 服务器地址*

ADDR = (**"127.0.0.1"**,8888)

*# 创建套接字*

sockfd = socket(AF\_INET,SOCK\_DGRAM)

*# 循环收发消息*

**while True**:

data = input(**"Msg>>"**)

**if not** data: *# 退出*

**break**

sockfd.sendto(data.encode(),ADDR)

msg,addr = sockfd.recvfrom(1024)

print(**"From server:"**,msg.decode())

sockfd.close()

1. 创建套接字

2. 收发消息

3. 关闭套接字

---------------

>总结 ：tcp套接字和udp套接字编程区别

>>1. 流式套接字是以字节流方式传输数据，数据报套接字以数据报形式传输

>>2. tcp套接字会有粘包，udp套接字有消息边界不会粘包

>>3. tcp套接字保证消息的完整性，udp套接字则不能

>>4. tcp套接字依赖listen accept建立连接才能收发消息，udp套接字则不需要

>>5. tcp套接字使用send，recv收发消息，udp套接字使用sendto，recvfrom

---------------------

### socket套接字属性

*"""*

*套接字属性介绍*

*"""*

**from** socket **import** \*

s = socket()

*# 设置端口立即重用*

s.setsockopt(SOL\_SOCKET,SO\_REUSEADDR,1)

s.bind((**'172.40.91.113'**,8989))

s.listen(3)

c,addr = s.accept()

print(**"地址类型:"**,s.family)

print(**"套接字类型:"**,s.type)

print(**"绑定地址:"**,s.getsockname())

print(**"文件描述符:"**,s.fileno())

*# 使用链接套接字调用*

print(**"获取连接端的地址:"**,c.getpeername())

c.recv(1024)

【1】 sockfd.type 套接字类型

【2】 sockfd.family 套接字地址类型

【3】 sockfd.getsockname() 获取套接字绑定地址

【4】 sockfd.fileno() 获取套接字的文件描述符

【5】 sockfd.getpeername() 获取连接套接字客户端地址

【6】 sockfd.setsockopt(level,option,value)

功能：设置套接字选项

参数： level 选项类别 SOL\_SOCKET

option 具体选项内容

value 选项值



## struct模块进行数据打包

\*\*\*代码实现：day6/struct\_recv.py\*\*\*

*"""*

*udp完成* *,从客户端输入学生信息,循环录入*

*学号:*

*姓名: 不会超过16字节*

*年龄:*

*分数: 保留一位小数*

*将信息发送给服务端,在服务端写入到一个文件里,*

*每个学生信息占一行*

*"""*

**from** socket **import** \*

**import** struct

*# 与客户端格式一直*

st = struct.Struct(**"i16sif"**)

*#udp套接字*

s = socket(AF\_INET,SOCK\_DGRAM)

s.bind((**'127.0.0.1'**,8888))

*# 打开一个保存信息的文件*

f = open(**'student.txt'**,**'a'**)

**while True**:

data,addr = s.recvfrom(1024)

*# (1,b'Lily',14,94.5)*

data = st.unpack(data)

*# 写入文件*

info = **"%d %-10s %d %.1f\n"**%data

f.write(info)

f.flush()

f.close()

s.close()

\*\*\*代码实现：day6/struct\_send.py\*\*\*

**import** struct

**from** socket **import** \*

*# 设置数据结构*

st = struct.Struct(**"i16sif"**)

*# 创建udp套接字*

s = socket(AF\_INET,SOCK\_DGRAM)

ADDR = (**'127.0.0.1'**,8888)

**while True**:

print(**"==========================="**)

id = int(input(**"学号:"**))

name = input(**"姓名:"**).encode()

age = int(input(**"年龄:"**))

score = float(input(**"得分:"**))

*# 将数据打包*

data = st.pack(id,name,age,score)

s.sendto(data,ADDR) *# 发送给服务端*

s.close()

1. 原理： 将一组简单数据进行打包，转换为bytes格式发送。或者将一组bytes格式数据，进行解析。

2. 接口使用

```python

Struct(fmt)

功能: 生成结构化对象

参数：fmt 定制的数据结构

st.pack(v1,v2,v3....)

功能: 将一组数据按照指定格式打包转换为bytes

参数：要打包的数据

返回值： bytes字节串

st.unpack(bytes\_data)

功能： 将bytes字节串按照指定的格式解析

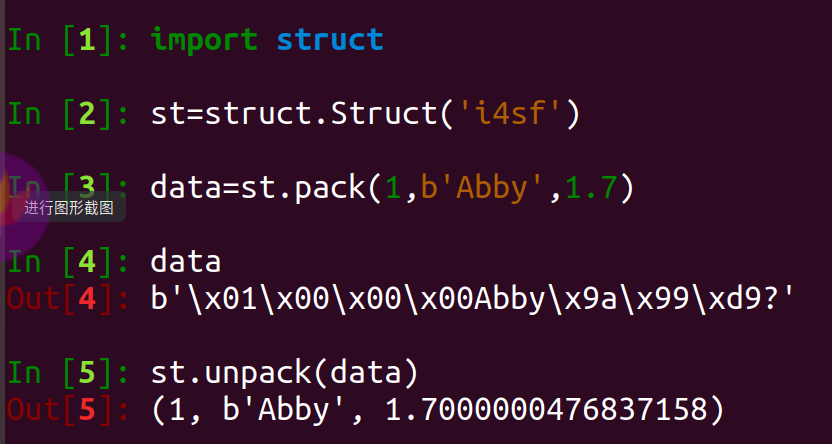
参数： 要解析的字节串

返回值： 解析后的内容

struct.pack(fmt,v1,v2,v3...)

struct.unpack(fmt,bytes\_data)

```



> 说明： 可以使用struct模块直接调用pack unpack。此时这两函数第一个参数传入fmt。其他用法功能相同

![](C:/Users/lvze/Desktop/%E8%AF%BE%E7%A8%8B%E4%B8%8B%E5%8F%91/IO%E7%BD%91%E7%BB%9C%E7%BC%96%E7%A8%8B/img/4\_struct.png)

### HTTP传输

#### HTTP协议 （超文本传输协议）

1. 用途 ： 网页获取，数据的传输

2. 特点

>\* 应用层协议，传输层使用tcp传输

>\* 简单，灵活，很多语言都有HTTP专门接口

>\* 无状态，协议不记录传输内容

>\* http1.1 支持持久连接，丰富了请求类型

3. 网页请求过程

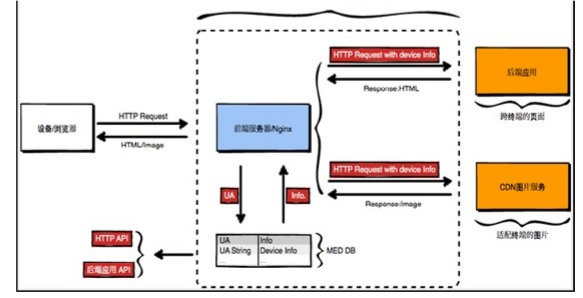
>1.客户端（浏览器）通过tcp传输，发送http请求给服务端

>2.服务端接收到http请求后进行解析

>3.服务端处理请求内容，组织响应内容

>4.服务端将响应内容以http响应格式发送给浏览器

>5.浏览器接收到响应内容，解析展示



#### HTTP请求（request）

\*\*\*代码实现：day6/http\_test.py\*\*\*

*"""*

*http请求响应演示*

*"""*

**from** socket **import** \*

*# tcp服务端*

s = socket()

s.setsockopt(SOL\_SOCKET,SO\_REUSEADDR,1) *# 端口立即重用*

s.bind((**'127.0.0.1'**,8000))

s.listen(5)

c,addr = s.accept()

print(**"Connect from"**,addr)

data = c.recv(4096).decode()

print(data) *# http请求*

html = **"""HTTP/1.1 200 OK**

**Content-Type: text/html**

**<h1>Hello World</h1>**

**"""**

c.send(html.encode())

c.close()

s.close()

\*\*\*代码实现：day7/http\_server.py\*\*\*

*"""*

*编写一个http服务端程序*

*如果浏览器的请求内容* */*

*响应码为* *200 OK,将index.html内容作为响应内容*

*如果浏览器的请求是其他的*

*响应码为* *404 Not Found 内容为* *"Sorry.."*

*"""*

**from** socket **import** \*

*# 与客户端交互*

**def** handle(connfd):

*# 获取http请求*

data = connfd.recv(4096).decode()

request\_line = data.split(**'\n'**)[0] *# 请求行*

info = request\_line.split(**' '**)[1] *# 请求内容*

*# 看一下请求内容是不是/*

**if** info == **'/'**:

**with** open(**'index.html'**) **as** f:

*# 组织http响应*

response = **"HTTP/1.1 200 OK\r\n"**

response += **"Content-Type:text/html\r\n"**

response += **'\r\n'**

response += f.read()

**else**:

response = **"HTTP/1.1 404 Not Found\r\n"**

response += **"Content-Type:text/html\r\n"**

response += **'\r\n'**

response += **"<h1>Sorry...</h1>"**

*# 发送给浏览器*

connfd.send(response.encode())

*# 搭建网络*

**def** main():

sockfd = socket()

sockfd.setsockopt(SOL\_SOCKET,SO\_REUSEADDR,1)

sockfd.bind((**'0.0.0.0'**,8000))

sockfd.listen(3)

**while True**:

connfd,addr = sockfd.accept()

print(**"Connect from"**,addr)

*# 处理客户端请求*

handle(connfd)

\* 请求行 ： 具体的请求类别和请求内容

```

GET / HTTP/1.1

请求类别 请求内容 协议版本

```

请求类别：每个请求类别表示要做不同的事情

```

GET : 获取网络资源

POST ：提交一定的信息，得到反馈

HEAD ： 只获取网络资源的响应头

PUT ： 更新服务器资源

DELETE ： 删除服务器资源

CONNECT

TRACE ： 测试

OPTIONS ： 获取服务器性能信息

```

\* 请求头：对请求的进一步解释和描述

```

Accept-Encoding: gzip

```

\* 空行

\* 请求体: 请求参数或者提交内容

#### http响应（response）

1. 响应格式：响应行，响应头，空行，响应体

\* 响应行 ： 反馈基本的响应情况

```

HTTP/1.1 200 OK

版本信息 响应码 附加信息

```

响应码 ：

```

1xx 提示信息，表示请求被接收

2xx 响应成功

3xx 响应需要进一步操作，重定向

4xx 客户端错误

5xx 服务器错误

```

\* 响应头：对响应内容的描述

```

Content-Type: text/html

```

\* 响应体：响应的主体内容信息