

# IO网络编程

Tedu Python 教学部
Author: 吕泽
Days: 4天

- ▼ IO网络编程
  - ▼ Linux操作系统及其组成
    - 1.操作系统的作用
    - 2.Linux操作系统组成
  - ▼ shell命令
    - 文件操作命令
  - IO
  - ▼ 文件
    - 字符串 (bytes)
    - 文件读写
    - ▼ 其它操作
      - 刷新缓冲区
      - 文件偏移量
      - 文件描述符
    - 文件管理函数
  - ▼ 网络编程基础
    - OSI七层模型
    - ▼ 四层模型 (TCP/IP模型)
      - 数据传输过程
      - 网络协议
    - 网络基础概念
  - ▼ 传输层服务
    - 面向连接的传输服务 (基于TCP协议的数据传输)
  - ▼ socket套接字编程
    - 套接字介绍
    - ▼ tcp套接字编程
      - tcp服务端流程
      - tcp客户端流程
      - tcp套接字数据传输特点
      - 网络收发缓冲区
      - tcp粘包
    - ▼ UDP套接字编程
      - udp服务端流程
      - udp客户端流程
    - ▼ socket套接字属性
      - UDP套接字广播
      - ▼ TCP套接字之HTTP传输
        - HTTP协议 (超文本传输协议)
        - HTTP请求 (request)
        - http响应 (response)
  - struct模块的使用

## Linux操作系统及其组成

### 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，并包括各种工具和应用软件。

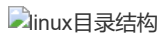
- Linux内核: Linux操作系统的核心代码
- 文件系统: 通常指称管理磁盘数据的系统，可将数据以目录或文件的型式存储。每个文件系统都有自己的特殊格式与功能
- shell命令: 接收用户命令，然后调用相应的应用程序，并根据用户输入的指令来反馈给用户指定的信息。



## shell命令

### 文件操作命令

- linux下的目录结构



作用	命令
切换工作目录	cd
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

- linux下凡是以点开头的文件或文件夹都是隐藏文件。
- linux的一条命令通常是由命令、选项、参数构成。

## IO

### 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，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。
- 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。

## 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。从功能角度分为文本文件(打开后会自动解码为字符)、二进制文件(视频、音频等)。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。

### 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制数据。因此在python3中字节串是常见的二进制数据展现方式。

- 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: `day1/file_open.py`  
代码实现: `day1/file_read.py`  
代码实现: `day1/file_write.py`

1. 打开文件

`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`'

文件模式	操作
<code>r</code>	以读方式打开 文件必须存在
<code>w</code>	以写方式打开 文件不存在则创建, 存在清空原有内容
<code>a</code>	以追加模式打开
<code>r+</code>	以读写模式打开 文件必须存在
<code>w+</code>	以读写模式打开文件 不存在则创建, 存在清空原有内容
<code>a+</code>	以读写模式打开 追加模式
<code>rb</code>	以二进制读模式打开 同 <code>r</code>
<code>wb</code>	以二进制写模式打开 同 <code>w</code>
<code>ab</code>	以二进制追加模式打开 同 <code>a</code>
<code>rb+</code>	以二进制读写模式打开 同 <code>r+</code>
<code>wb+</code>	以二进制读写模式打开 同 <code>w+</code>
<code>ab+</code>	以二进制读写模式打开 同 <code>a+</code>

`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。

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。

1.读取文件

- `read([size])`  
功能: 用来直接读取文件中的字符。  
参数: 如果没有给定`size`参数 (默认值为-1) 或者`size`值为负, 文件将被读取直到末尾, 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末尾, 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`表示最多读取指定的字符 (字节)。  
返回值: 返回读取到的内容
- `readlines([sizeint])`  
功能: 用来读取文件中的每一行作为列表的一项。  
参数: 如果没有给定`size`参数 (默认值为-1) 或者`size`值为负, 文件将被读取直到末尾, 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。

```
for line in f:
    print(line)
```

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

- 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

#### 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()
```

#### 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：

```
with context_expression [as target(s)]:  
    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。

```
with open("file", "r+") as f:  
    f.read()
```

注意

- 1.加b的打开方式读写要求必须都是字节串
- 2.无论什么缓冲，当程序结束或者文件被关闭时，都会将缓冲区内容写入磁盘

## 其它操作

### 刷新缓冲区

**代码实现：** day02/buffer.py

```
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：** day02/seek.py

#### 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z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。

- 必须以二进制方式打开文件时基准位置只能是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)

## 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:提供用户服务, 具体功能有应用程序实现

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

网络层: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-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）

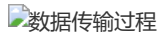
背景：实际工作中工程师无法完全按照七层模型要求操作，逐渐演化为更符合实际情况的四层

OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	TFTP,FTP,NFS,WAIS
表示层（Presentation）	应用层	Telnet,Rlogin,SNMP,Gopher
会话层（Session）	应用层	SMTP,DNS
传输层（Transport）	传输层	TCP,UDP
网络层（Network）	网际层	IP,ICMP,ARP,RARP,AKP,UUCP
数据链路层（Data Link）	网络接口层	FDDI,Ethernet,Arpanet,PDN,SLIP,PPP
物理层（Physics）	网络接口层	IEEE 802.1A,IEEE 802.2到IEEE 802.11

### 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
- IPV6 128位 扩大了地址范围

- 域名
- 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
- 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，通信结束要断开连接。
- 3. 适用情况：对数据传输准确性有明确要求，传输文件较大，需要确保可靠性的情况。比如：网页获取，文件下载，邮件收发。

- 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



- 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
- 主动方收到确认，发送最终报文完成断开



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，数据手法自由随意。
2. 使用情况：网络较差，对传输可靠性要求不高。比如：网络视频，群聊，广播

### 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？
- 三次握手和四次挥手指什么，过程是怎样的？

## 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计数手段
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）：以数据包形式传输数据，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

#### tcp服务端流程

 tcp套接字服务端流程

**代码实现：day02/tcp\_server.py**

#### 1.创建套接字

```
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（只有有子协议的套接字才会选择，例如底层套接字，但是最常用的TCP和UDP套接字没有子协议，所以为0）


返回值：套接字对象

#### 2.绑定地址

本地地址："localhost","127.0.0.1"

网络地址："172.40.91.185"

自动获取地址："0.0.0.0"

 tcp套接字bind

```
socket.bind(addr)
```

功能：绑定本机网络地址

参数：二元元组(ip, port)("0.0.0.0",8888)

#### 3.设置监听

```
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

```
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

```
data = connfd.recv(bufsize)
```

功能：接受客户端消息

参数：每次最多接受消息的大小

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

```
sockfd.close()
```

功能：关闭套接字

## tcp客户端流程

**代码实现：** `day02/tcp_client.py`

 tcp套接字客户端流程

### 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

```
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连接中当一段推出，另一端如果阻塞在recv，此时recv会立即返回一个空字节串。
- tcp连接中如果一段已经不存在，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

- 网络缓冲区有效的协调了消息的收发速度
- 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

**代码实例：** `day03/stick_send.py, stick_recv.py`

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受，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：

- 人为的添加消息边界
- 控制发送速度

## UDP套接字编程

### udp服务端流程

 udp套接字服务端流程

**代码实现：** `day02/udp_server.py`

#### 1.创建数据报套接字



```
sockfd = socket(AF_INET, SOCK_DGRAM)
```

2.绑定地址

```
sockfd.bind(addr)
```

3.消息收发

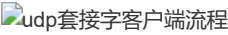
```
data, addr = sockfd.recvfrom(bufferSize)
功能：接收UDP消息
参数：每次最多接收多少字节
返回值：data 接收到的内容
        addr 消息发送方地址

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

```
sockfd.close()
```

udp客户端流程



代码实现

- 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，数据包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

代码实现： day03/sock\_attr.py

- 【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，可以传递的参数时dir(socket)成员中，以SOL\_、IPPROTO\_、开头的。现阶段常用的是SOL\_SOCKET
option具体选项内容，可以传递的参数时dir(socket)成员中，以IPV\_、IP\_、SO\_、TCP\_开头的。
value选项值
- 【7】sockfd.getsockopt(level, option)
功能：获取套接字选项值

选项	意义	期望值
SO_BINDTODEVICE	可以使socket只在某个特殊的网络接口（有效）。也许不能是移动便携设备	一个字符串给出设备的名称或者一个空字符串返回默认值

选项	意义	期望值
SO_BROADCAST	允许广播地址发送和接受信息包。只对UDP有效。 如何发送和接收广播信息包	布尔型整数
SO_DONTROUTE	禁止通过路由器和网关向外发送信息包。 这主要是为了安全而用在以太网上UDP通信的一种方法。 不管目的地址使用什么IP地址，都可以防止数据离开本地网络	布尔型整数
SO_KEEPAIVE	可以使TCP通信的信息包保持连续性。 这些信息报可以在没有信息传输的时候， 使通信的双方确定连接时保持的	布尔型整数
SO_OOINLINE	可以把收到的不正常数据看成是正常的数 据，也就是说会通过一个标准的对recv()的调用来接收这些数据	布尔型整数
SO_REUSEADDR	当socket关闭后， 本地端用于该socket的端口号立刻就可以被重用。通常来说， 只有经过系统定义一段时间后，才能被重用。	布尔型整数

UDP套接字广播

\*\*\*代码实现： day03/broadcast\_send.py

\*\*\*代码实现： day03/broadcast\_recv.py

- 广播定义：一端发送多点接收
- 广播地址：每个网络的最大地址为发送广播的地址，向该地址发送，则网段内所有主机都能接收。

TCP套接字之HTTP传输

HTTP协议（超文本传输协议）

1.用途：网业获取，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，组织相应内容
- 4.服务端将相应内容以http相应格式发送给浏览器
- 5.浏览器接收到相应内容，解析展示



HTTP请求（request）

格式：请求行，请求头，空行，请求体

代码实现： day04/http\_test.py

代码实现： day04/http\_server.py

- 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：使用键值对，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)

响应格式：响应行、响应头、空行、响应体

- 响应行：反馈基本的相应情况

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：响应的主题内容信息

## struct模块的使用

**代码实现：** day04/struct\_recv.py  
**代码实现：** day04/struct\_send.py

- 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。
- 接口使用

struct.Struct(fmt)  
功能：生成结构化对象  
参数：fmt 定制的数据结构  
例： 1 lisa 1.75的fmt为i4sf, i表示1转成整形，4s表示有4个字符转成字符串，f表示1.75的浮点数

st.pack(v1, v2, v3...)  
功能：将一组数据按照指定格式打包转换为bytes，函数的实参如果要传递字符，需要传递字节串，例如不能传递"Lisa"，而必须是b"Lisa"，  
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成一个元组返回  
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。此时这两个函数第一个参数传入fmt。其他用法功能相同。

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)

Format	C Type	Python type	Standard size	Notes
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
l	unsigned short	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2),(3)
Q	unsigned long long	integer	8	(2),(3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
e	(7)	float	2	(5)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes	1	
P	void *	long		