# **IO网络编程**

| **Tedu Python 教学部** |
| --- |
| Author：吕泽 |
| Days：3天 |

* [IO](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "io)
* [文件](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "文件)
  + [字节串（](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "字节串bytes)bytes）
  + [文件读写](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "文件读写)
    - [@](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "扩展延伸文件流) 扩展延伸：文件流
  + [其他操作](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "其他操作)
    - [刷新缓冲区](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "刷新缓冲区)
    - [文件偏移量](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "文件偏移量)
    - [文件描述符](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "文件描述符)
  + [文件管理函数](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "文件管理函数)
* [网络编程基础](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "网络编程基础)
  + [OSI](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "osi七层模型)七层模型
  + [四层模型（](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "四层模型tcpip模型)TCP/IP模型）
    - [数据传输过程](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "数据传输过程)
    - [网络协议](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "网络协议)
  + [网络基础概念](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "网络基础概念)
* [传输层服务](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "传输层服务)
  + [面向连接的传输服务（基于](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "面向连接的传输服务基于tcp协议的数据传输)TCP协议的数据传输）
  + [面向无连接的传输服务（基于](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "面向无连接的传输服务基于udp协议的数据传输)UDP协议的数据传输）
* [socket](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "socket套接字编程)套接字编程
  + [套接字介绍](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "套接字介绍)
  + [tcp](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "tcp套接字编程)套接字编程
    - [服务端流程](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "服务端流程)
    - [客户端流程](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "客户端流程)
    - [tcp](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "tcp-套接字数据传输特点) 套接字数据传输特点
    - [网络收发缓冲区](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "网络收发缓冲区)
    - [tcp](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "tcp粘包)粘包
  + [UDP](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "udp套接字编程)套接字编程
    - [服务端流程](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "服务端流程-1)
    - [客户端流程](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "客户端流程-1)
  + [socket](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "socket模块方法和socket套接字属性)模块方法和socket套接字属性
  + [UDP](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "udp套接字广播)套接字广播
  + [TCP](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "tcp套接字之http传输)套接字之HTTP传输
    - [HTTP](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "http协议-超文本传输协议)协议 （超文本传输协议）
    - [HTTP](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "http请求)请求
    - [http](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "http响应response)响应（response）
* [struct](http://code.tarena.com.cn/AIDCode/aid1903/02-PythonNet/Net/pythonNet.html" \l "struct模块的使用)模块的使用

## **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

1. 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程数据。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 ：encode()
* 字节串转换为字符串方法 : decode()

代码实现： day1/code.py

s = input(**">>"**)

*#　字符串* *--> 字节串*

byte = s.encode()

print(**"bytes:"**,byte)

*#　字节串* *--> 字符串*

s = byte.decode()

print(**"str:"**,s)

*#　打印字节串*

print(**"bytes:"**,**b"hello world"**)

*# 通过字节串转换函数转换为字节串*

print(**"int :"**,bytes(**'abc'**,encoding=**'utf-8'**))

### **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

*#　文件操作示例*

*# 打开文件*

**try**:

fd = open(**"test"**,**'a'**)

**except** FileNotFoundError **as** e:

print(e)

**else**:

print(**"打开文件成功"**)

*# 开始你的读写操作*

*# 关闭文件*

fd.close()

代码实现： day1/file\_read.py

**import** time

*# 1.打开文件返回文件对象*

fd=open(**'test'**,**'r'**)

*# 2.读操作*

*# read()*

*# 如果不加参数读到文件末尾，如果文件过大时不建议这样做，会消耗大量内存，读取时间也会很长*

*# 如果读取到文件末尾再读取会得到空字符串*

*# 实际读取到的内容可能小于或等与size的大小*

*# data=fd.read(13)*

*# print(data)*

*# n=1*

*# while n<20:*

*# data=fd.read(n)*

*# print("\r%s"%data,end='')*

*# n+=1*

*# time.sleep(1)*

*# readline（）*

*#*

*# data0=fd.readline() # 读取一行*

*# data1=fd.readline(4)*

*# data2=fd.readline() # 接着上一行继续读取*

*# print(data2)*

*# while True:*

*# data=fd.readline()*

*# if not data:*

*# break*

*# print(data,end='')*

*# 将读取内容作为列表返回*

*# data=fd.readlines(15)*

*# print(data)*

*# fd为可迭代对象，使用for循环每次可获得一行内容*

n=1

**for** line **in** fd:

print(type(line))

print(**"第%d行内容：%s"**%(n,line))

**for** i **in** range(len(line)):

**if** line[i] == **'.'**:

n = int(line[:i])

print(n+1)

*# print(type(line))*

n+=1

*# 写操作*

*# 3.关闭文件*

fd.close()

代码实现： day1/file\_write.py

*# 写操作*

fd =open(**"test"**,**"a"**)

*# 1. write(string)*

*# 如果想换行需要人为添加换行符*

*# 注意：*

*# 如果open打开文件的方式中有b，表示以二进制打开，此时读写操作得到的*

fd.write(**"hello 死鬼\n"**)

fd.write(**"干啥呀"**)

*# 循环写入*

*# while True:*

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

*# if data=='quit':*

*# break*

*# fd.write("%s\n"%data)*

*# 2. writelines(string)*

*# 写入一个列表内容*

*# fd.writelines(['hello\n','world\n'])*

fd.close()

#### **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

r 以读方式打开 文件必须存在

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

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————参数0表示无缓冲，1表示有行缓冲，如果是大于1标识直接指明缓冲区大小。如果不写或为负数则表示使用系统默认提供的缓冲机制。

**返回值**：成功返回文件流对象。失败得到IOError。

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，装满缓冲区在一起送到磁盘中去。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，然后再从缓冲区逐个的将数据送到程序的数据区。

代码事例： day02/buffer.py

*# 缓冲区*

fd=open(**'test'**,**'w'**) *# 系统默认的缓冲区*

*# fd=open('test','w',0) # 无缓冲（不允许）*

*# fd=open('test','w',1) # 行缓冲*

*# fd=open('test','w',12) # 指明缓冲区大小（不识别）*

**while True**:

s=input(**">>>"**)

**if** s==**'quit'**:

**break**

fd.write(s)

fd.flush() *# 立即刷新缓冲区并将内容写入磁盘*

fd.close()

* + - 1. @ 扩展延伸：文件流

代码实现： day1/stdin.py

流（stream）：所有的I/O操作仅是简单的从程序移进或者移出，这种字节流，就称为流。所以打开一个文件有时也称为打开一个文件流对象

系统已经默认为我们打开了三个流，在sys模块中：

标准输入————sys.stdin

标准输出————sys.stdout

标准错我————sys.stderr

#### ２．读取文件

**read**([size])  
size 表示字符数，用来直接读取字节到字符串中，最多读取给定数目个字节。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。

１．文件过大时候建议在non-blocking模式下使用，因为会消耗大量内存，读取时间也会更长。

２．如果读取到文件结尾再读取会得到空字符串

３．实际读取到的内容可能小于等于指定 size大小

**breadline**([size])  
读取打开文件的一行(读取下个行结束符之前的所有字节)。然后整行，包括行结束符，作为字符串返回。和 read() 相同，它也有一个可选的 size 参数，默认为 -1，代表读至行结束符。如果提供了该参数，那么在超过size个字节后会返回不完整的行。

１．每次读取一行内容遇到换行终止

２．如果指定 size，则只读取 size大小的内容

**readlines**([sizeint])  
该方法并不像其它两个输入方法一样返回一个字符串。它会读取所有(剩余的)行然后把它们作为一个字符串列表返回。它的可选参数sizhint代表返回的最大字节大小。

文件对象本身也是一个迭代器，在for循环中可以迭代文件的每一行。

１．读取到整行内容作为列表返回，每行是列表中的一个元素

２．

for line in f:

print(line)

#### ３．写入文件

**write**(string)  
功能与 read() 和 readline() 相反。它把含有文本数据或二进制数据块的字符串写入到文件中去。

１．如果想换行需要人为的添加换行符(\n)

**writelines**(str\_list)  
和 readlines() 一样，writelines()方法是针对列表的操作，它接受一个字符串列表作为参数，将它们写入文件。行结束符并不会被自动加入，所以如果需要的话，你必须在调用writelines()前给每行结尾加上行结束符。

**注意：**如果open打开文件的方式中有b，表示以二进制打开，此时读写操作得到或者

#### 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：

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()

### **其他操作**

#### **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(人为刷新缓冲区)，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

*# 文件偏移量示例*

fd = open(**'test'**,**'r+'**)

*#　相对开头位置向后偏移了多少*

print(**"当前文件偏移量位置："**,fd.tell()) *#0*

print(fd.read(2))

print(**"当前文件偏移量位置："**,fd.tell()) *#2*

*#　人为调整文件偏移*

fd.seek(5,0) *#　相对开头位置向后偏移5个*

print(fd.read(2))

fd.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。

1. 基本操作

tell()  
功能：获取文件偏移量

**seek**(offset[,whence])  
功能:移动文件为止  
参数：offset字节代表相对于某个位置偏移量。可以是负数表示向前移动。  
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。

#### **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。

1. 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)

1. 查看文件列表

os.listdir(dir)

1. 查看文件是否存在

os.path.exists(file)

1. 判断文件类型

os.path.isfile(file)

1. 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**

* 网络主机（host）

功能：标识一台主机在网络中的位置（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'

查看本机网络地址命令： ifconfig

* IP地址

功能：确定一台主机的网络路由位置

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

特殊IP

127.0.0.1 本机测试IP  
0.0.0.0 自动获取本机网卡地址  
172.40.91.0 通常表示一个网段  
172.40.91.1 通常表示一个网关  
172.40.91.255 用作广播地址

* 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（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

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

1. 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，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：day2/tcp\_server.py

*"""*

*TCP套接字服务端*

*重点代码*

*"""*

**import** socket

**import** struct

*# 1.创建套接字*

sockfd=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

*# 2.绑定地址*

sockfd.bind((**'176.140.6.128'**,8888))

*# 3.设置监听*

sockfd.listen(5)

*# 4.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:

print(**"Exit Server"**)

**break**

*# 5.消息收发*

*# while True:*

data = connfd.recv(1024) *# recv()阻塞函数*

data=struct.unpack(**'i4sf'**,data)

*# if not data:*

*# break*

print(**"接收到的消息："**,data)

*# message=input("请输入发送的消息")*

n = connfd.send(**"Please input your message"**.encode())

*# print("发送了%d个字节的数据"%n)*

*# 6.关闭套接字*

connfd.close()

sockfd.close()

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

返回值： 套接字对象

1. 绑定地址

sockfd.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)

1. 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

1. 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

1. 消息收发

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

1. 关闭套接字

sockfd.close()

功能：关闭套接字

#### **客户端流程**

代码实现：day2/tcp\_client.py

*"""*

*TCP套接字客户端*

*重点代码*

*"""*

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

**import** struct

*# 1.创建套接字*

sockfd=socket()

*# 2.请求连接*

server\_addr=(**'176.140.6.128'**,8002)

sockfd.connect(server\_addr)

*# 3.收发消息*

**while True**:

message=input(**"输入发送的信息："**)

*# message=struct.pack('i4sf',1,b'lili',99.5)*

**if not** message:

**break**

sockfd.send(message.encode())

data=sockfd.recv(1024)

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

*# 4.关闭套接字*

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

1. 请求连接

sockfd.connect(server\_addr)

功能：连接服务器

参数：元组 服务器地址

1. 收发消息

注意： 防止两端都阻塞，recv send要配合

1. 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，让然试图通过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

### **UDP套接字编程**

#### **服务端流程**

代码实现：day2/udp\_server.py

*"""*

*UDP套接字服务端*

*重点代码*

*"""*

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

*# 1.创建数据报套接字*

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

*# 2.绑定地址*

addr=(**'176.140.6.128'**,8888)

sockfd.bind(addr)

*# 3. 消息收发*

**while True**:

**try**:

data,addr = sockfd.recvfrom(5)

**except** KeyboardInterrupt:

print(**"Exit Server"**)

**break**

*# if not data:*

*# break*

print(**"收到的消息："**,data.decode())

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

*# 4. 关闭套接字*

sockfd.close()

1. 创建数据报套接字

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

1. 绑定地址

sockfd.bind(addr)

1. 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

1. 关闭套接字

sockfd.close()

#### **客户端流程**

代码实现：day2/udp\_client.py

*"""*

*UDP套接字客户端*

*重点代码*

*"""*

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

*# 1. 创建套接字*

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

*# 2. 收发消息*

**while True**:

data=input(**"发送："**)

**if not** data:

**break**

addr=(**'176.140.6.128'**,8888)

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

data,add=sockfd.recvfrom(1024)

print(data.decode())

*# 3. 关闭套接字*

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模块方法和socket套接字属性**

1. 部分socket模块方法

【1】 gethostname() 获取计算机名

【2】 gethostbyname('www.baidu.com') 获取主机ip地址

【3】 getservbyname('mysql') 获取服务端口号

【4】 getservbyport(3306) 获取端口对应服务

【5】 inet\_aton('192.168.1.2') 将IP转换为bytes子串

【6】 inet\_ntoa(b'\xc0\xa8\x01\x02') 将bytes子串转换为IP地址

1. 套接字属性  
   代码实现：day2/sock\_attr.py

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

*# 创建一个tcp套接字*

s=socket()

*# 设置套接端口立即使用*

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

s.bind((**"176.140.6.128"**,8888))

s.listen(5)

c,addr = s.accept()

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

c.recvfrom(1024)

print(**"地址类型"**,s.family)

print(**"套接字类型"**,s.type)

print(**"获取套接字绑定地址"**,s.getsockname())

print(**"描述符"**,s.fileno())

**try**:

print(**"获取连接套接字客户端地址"**,c.getpeername())

**except** OSError:

print(**"客户端不存在"**)

s.close()

【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

【7】 sockfd.getsockopt(level,option)

功能 ： 获取套接字选项值

### **UDP套接字广播**

代码实现：day2/broadcast\_recv.py

*# 广播接受*

*"""*

*1. 创建ｕｄｐ套接字*

*2. 设置套接字为可以接收广播*

*3. 选择接收端口*

*"""*

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

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

*#　让套接字可以接收广播*

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

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

**while True**:

**try**:

msg,addr = s.recvfrom(1024)

**except** KeyboardInterrupt:

**break**

**else**:

print(msg.decode())

s.close()

代码实现：day2/broadcast\_send.py

*#　广播发送*

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

**from** time **import** sleep

*# 广播地址*

dest = (**'172.40.91.255'**,9999)

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

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

data = **"""**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**5.8 北京　　初夏**

**喜欢夏天，但你比夏天更加明媚**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**"""**

**while True**:

sleep(2)

s.sendto(data.encode(),dest)

s.close()

* 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

1. 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**

代码实现：day3/http\_test.py

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

*#　创建ｔｃｐ套接字*

s = socket()

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

s.listen(3)

c,addr = s.accept()

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

data = c.recv(4096)

print(data)

*#　ｈｔｔｐ响应格式*

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

**Content-Type:text/html**

**<h1>hello world</h1>**

**"""**

c.send(data.encode())

c.close()

s.close()

代码实现：day3/http\_server.py

*"""*

*http 功能演示*

*将网页发送给浏览器展示*

*"""*

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

*# 　处理浏览器的http请求*

**def** handle(connfd):

print(**"Request from"**, connfd.getpeername())

request = connfd.recv(4096) *# 接收http请求*

*# 　防止客户端断开*

**if not** request:

**return**

*# 将request按行分割*

request\_line = request.splitlines()[0].decode()

*# 　获取请求内容*

info = request\_line.split(**' '**)[1]

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

f = open(**'index.html'**)

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>"**

*# 向浏览器发送内容*

connfd.send(response.encode())

*# 　搭建ｔｃｐ网络*

**def** main():

sockfd = socket()

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

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

sockfd.listen(3)

print(**"Listen the port 8000..."**)

**while True**:

connfd, addr = sockfd.accept()

handle(connfd) *# 处理浏览器请求*

connfd.close()

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

main()

* 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

## **struct模块的使用**

代码实现：day3/struct\_recv.py

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

**import** struct

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

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

*# 　确定数据结构*

st = struct.Struct(**"i32sif"**)

*# 　打开文件*

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

**while True**:

data, addr = s.recvfrom(1024)

*# 　数据解析*

data = st.unpack(data)

*# 　整理写入内容*

info = **"%d %s %d %.2f\n"**%(\

data[0],data[1].decode(), data[2], data[3])

f.write(info)

f.close()

s.close()

代码实现：day3/struct\_send.py

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

**import** struct

*#　接收端地址*

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

*#　规定数据格式*

st = struct.Struct(**'i32sif'**)

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

**while True**:

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

id = int(input(**"ID:"**))

name = input(**"NAME:"**).encode()

age = int(input(**"AGE:"**))

score = float(input(**"SCORE:"**))

*#　数据打包*

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

s.sendto(data,ADDR)

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

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

# 练习

## 每日课堂笔记

### 五月五日(day01)

#### １．字节串＆字符串

*"""*

*重点代码*

*"""*

s = input(**">>"**)

print(**"str:"**,s)*# 字符串*

print(**"bytes:"**,**b"hello world"**)*# 字节串*

byte=s.encode()

print(**"字符串转换字节串:"**,byte)*# 字符串---->字节串*

str=byte.decode()

print(**"字节串转换字符串:"**,str)*# 字节串---->字符串*

*# 通过字节串转换函数转换为字节串*

print(**"int i:"**,bytes([1,2,3]))

*#bytes()*

*# 参数：*

*# 整数ｎ 初始化一个长度为n的列表序列*

*# 字符串　　将字符串转换为字节串*

*# 0-255整数列表　　将列表整数转换bytes字节串*