NSD Python2 DAY04

1 案例1：分析apache访问日志

1.1 问题

编写count_patt.py脚本，实现一个apche日志分析脚本：

1. 统计每个客户端访问apache服务器的次数
2. 将统计信息通过字典的方式显示出来
3. 分别统计客户端是Firefox和MSIE的访问次数
4. 分别使用函数式编程和面向对象编程的方式实现

1.2 方案

collections是python内建的一个集合模块，模块中提供了许多有用的集合类,其中counter类是一个简单的计数器，以字典的键值对形式储存，其中搜索的元素作为键，出现的次数作为值

实现过程：

1.实例化一个计数器

2.实例化正则表达式

3.将文件以对象形式打开

4.通过正则表达式查找文件每一行

5.如果找到结果

6.将结果添加到计数器，通过update方法更新原有数据

7.返回计数器

8.将文件地址和正则表达式作为实参传递给函数

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day08]# vim count_patt.py
#! /usr/bin/env python3

import re
import collections

```

Top

```
07.     #fname 文件地址  patt 正则表达式
08.     def count_patt(fname,patt):
09.
10.         counter = collections.Counter()
11.
12.         cpatt = re.compile(patt)
13.             with open(fname) as fobj:
14.             for line in fobj:
15.
16.                 m = cpatt.search(line)
17.
18.                 if m:
19.
20.                     counter.update([m.group()])
21.
22.         return counter
23.
24.     if __name__ == "__main__":
25.         fname = "access_log.txt"
26.         ip_patt = "^(\d+\.){3}\d+"
27.         a = count_patt(fname,ip_patt)
28.         print(a)
29.         br_patt = "Firefox|MSIE|Chrome"
30.         b = count_patt(fname,br_patt)
31.         print(b)
```

实现此案例还可通过面向对象方式实现：

实现过程：

1.创建类CountPatt()

2.定义构造方法 创建正则对象

3.定义类方法

4.创建计数器对象

5.打开文本文件

6.通过正则表达式查找文件每一行

7.如果找到结果

8.将结果添加到计数器，通过update方法更新原有数据

9.返回计数器

10.将文件地址和正则表达式作为实参传递给函数 Top

```
[root@localhost day08]# vim count_patt2.py
#! /usr/bin/env python3

import re
import collections


import re
import collections


class CountPatt(object):

    def __init__(self,patt):
        self.cpatt = re.compile(patt)


    def count_patt(self,fname):

        counter = collections.Counter()

        with open(fname) as fobj:

            for line in fobj:

                m = self.cpatt.search(line)

                if m:

                    counter.update([m.group()])

        return counter

if __name__ == "__main__":
    fname = "access_log.txt"
    ip_patt = "^(\d+\.){3}\d+"
    br_patt = "Firefox|MSIE|Chrome"
    ip = CountPatt(ip_patt)
    print(ip.count_patt(fname))
    br = CountPatt(br_patt)
```

Top

```
41.         print(br.count_patt(fname))
```

步骤二：测试脚本执行

```
01.     [root@localhost day08]# python3 count_patt.py
02.     Counter({'172.40.0.54': 391, '172.40.50.116': 244, '201.1.1.254': 173, '127
03.     Counter({'Firefox': 870, 'MSIE': 391, 'Chrome': 24})
04.     [root@localhost day08]# python3 count_patt2.py
05.     Counter({'172.40.0.54': 391, '172.40.50.116': 244, '201.1.1.254': 173, '127
06.     Counter({'Firefox': 870, 'MSIE': 391, 'Chrome': 24})
```

2 案例2：创建TCP时间戳服务器

2.1 问题

创建tcp_time_serv.py脚本，要求编写一个TCP服务器：

1. 服务器监听在0.0.0.0的21567端口上
2. 收到客户端数据后，将其加上时间戳后回送给客户端
3. 如果客户端发过来的字符全是空白字符，则终止与客户端的连接

2.2 方案

服务器进程首先要绑定一个端口并监听来自其他客户端的连接。如果某个客户端连接过来了，服务器就与该客户端建立Socket连接，随后的通信就靠这个Socket连接了.

服务器需要有自己的地址和端口，并且还需要获取客户端地址和端口，同时需要不断的监听客户端的链接。

服务器端流程如下：

1.建立socket对象

2.设置socket选项，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。

3.绑定socket。即为服务器要求一个端口号。

4.将套接字设为监听模式，准备接收客户端请求。利用listen()函数进行侦听连接。该函数只有一个参数，其指明了在服务器实际处理连接的时候，允许有多少个等待的连接在队列中等待。作为一个约定，很多人设置为5。如：s.listen(5)

5.accept()会等待并返回一个客户端的连接

6.用返回的套接字和客户端进行通信，分别使用send和recv函数接收和发送数据

7.关闭套接字，当服务器关闭时要关闭所有的套接字，和释放资源。

2.3 步骤

Top

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day08]# vim tcp_time_serv.py

#! /usr/bin/env python3

import socket
import time

host = '0.0.0.0'
port = 21567
addr = (host, port)
#第一步：建立socket对象
s = socket.socket()
#第二步：设置socket选项
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)        #这里value设置为1，表
#第三步：绑定socket
s.bind(addr)
#第四步：侦听连接。
s.listen(2) #相当于有多少个客户端可以同时发送过来数据
#第五步：接受一个新连接
cli_sock, cli_addr = s.accept()
print('Client connected from:', cli_addr)
#第六步：用返回的套接字和客户端进行通信，接收和发送数据
# 将收到的bytes类型，转成utf8字符串
data = str(cli_sock.recv(1024), encoding='utf8')
data = '[%s] %s' %(time.strftime('%H:%M:%d'), data)
print(data)
# 发送时，将utf8字符串转成bytes类型
cli_sock.sendall(bytes(data, encoding='utf8'))
#第七步：关闭套接字
cli_sock.close()
s.close()
```

实现此案例还可用以下方法：

accept函数是放在一个死循环中的，一直监听客户的请求

将send和recv函数放在死循环中持续发送和接收数据

Top

```
[root@localhost day08]# vim tcp_time_serv2.py

```

```
#! /usr/bin/env python3

import socket
import time

host = '0.0.0.0'
port = 21567
addr = (host, port)
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(addr)
s.listen(2)

while True:
    try:
        cli_sock, cli_addr = s.accept()
       #捕获用户终端执行异常，终止连接
    except KeyboardInterrupt:
        break

    print('Client connected from:', cli_addr)
    while True:
        data = str(cli_sock.recv(1024), encoding='utf8')
        print(data)
# if data.strip() == '':
        if not data.strip():
            break
        data = '[%s] %s' %(time.strftime('%H:%M:%d'), data)
        print(data)
cli_sock.sendall(bytes(data, encoding='utf8'))
    cli_sock.close()
s.close()
```

实现此案例还可用以下方法：利用类方法实现代码

```
[root@localhost day08]# vim tcp_time_serv3.py

#! /usr/bin/env python3

import socket
```

Top

```
import time

class TcpTimeServ:
    def __init__(self, host, port):
        self.addr = (host, port)
        self.serv = socket.socket()
        self.serv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.serv.bind(self.addr)
        self.serv.listen(2)

    def handle_child(self, cli_sock):
        while True:
            data = str(cli_sock.recv(1024), encoding='utf8')
            print(data)
            if not data.strip():
                break
            data = '[%s] %s' %(time.strftime('%H:%M:%d'), data)
            print(data)
            cli_sock.sendall(bytes(data, encoding='utf8'))
        cli_sock.close()

    def mainloop(self):
        while True:
            cli_sock, cli_addr = self.serv.accept()
            self.handle_child(cli_sock)
            cli_sock.close()
        self.serv.close()

if __name__ == '__main__':
    s = TcpTimeServ('0.0.0.0', 21567)
    s.mainloop()
```

步骤二：测试脚本执行

```
[root@localhost day08]# python3 tcp_time_serv3.py
nihao
[14:20:25] nihao
wohenhao
[14:20:25] wohenhao
```

Top

3 案例3：创建TCP时间戳客户端

3.1 问题

创建tcp_time_client.py文件，编写一个TCP客户端：

1. 连接服务器的21567
2. 接收用户从键盘上的输入
3. 发送接收到的字符串给服务器
4. 如果用户按ctrl + c则退出程序

3.2 方案

运行服务器端，让服务器端处于等待状态，运行TCP客户端的同时指定服务器ip地址与端口，然后输入信息，回车后会得到服务器返回信息，然后等待服务器向其发送信息后退出。

客户端的流程如下：

1.创建一个套接字（socket）

2.向服务器发出连接请求（connect），值得注意的是，客户端要主动发起TCP连接，必须知道服务器的IP地址和端口号

3.和服务器端进行通信（send/recv）

4.关闭套接字

需要注意的是：接收数据时，调用recv(max)方法，一次最多接收指定的字节数，因此，在一个while循环中反复接收，直到recv()返回空数据，表示接收完毕，退出循环。

当我们接收完数据后，调用close()方法关闭Socket，这样，一次完整的网络通信就结束了

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day08] # vim tcp_time_client.py
#! /usr/bin/env python3

import socket
import sys

class TcpTimeClient:
    def __init__(self, host, port):
        self.addr = (host, port)
#创建套接字
        self.cli = socket.socket()
#建立连接
        self.cli.connect(self.addr)

    def chat(self):
```

Top

```
16.             while True:
17.                 data = input( '>' )
18.                 #发送数据
19.                 self.cli.sendall( data)
20.                 if not data:
21.                     break
22.                  #接收数据
23.                 print( str( self.cli.recv( 1024) , encoding='utf8' ) )
24.             self.cli.close( )
25.
26.     if __name__ == '__main__':
27.         c = TcpTimeClient( sys.argv[ 1] , int( sys.argv[ 2] ) )
28.         c.chat( )
```

步骤二：测试脚本执行

```
01.     [ root@localhost day08] # python3 tcp_time_client.py  '0.0.0.0' 21567
02.     >: 'nihao'
03.     nihao
04.     [ 14: 20: 25]  nihao
05.     >: 'wohenhao'
06.     wohenhao
07.     [ 14: 20: 25]  wohenhao
```

4 案例4：创建UDP时间戳服务器

4.1 问题

创建udp_time_serv.py脚本，编写一个UDP服务器：

1. 服务器监听在0.0.0.0的21567端口上
2. 收到客户端数据后，将其加上时间戳后回送给客户端

4.2 方案

UDP的通信与TCP相类似，使用UDP的通信双方也分为客户端和服务器，服务器首先需要创建Socket对象，设置socket选项，服务器要绑定一个端口接收来自客户端的数据，并向该客户端发送数据。

需要注意的是：

1.UDP则是面向无连接的协议。只要数据发送出去。无需去管对方是否接收到。使用UDP协议时，不需要建立连接，只需要知道对方的IP地址和端口号，就可以直接发数据包。但是，能不能到达就无法确定了。

2.虽然用UDP传输数据不可靠，但它的优点是和TCP比，速度快，对于不要求可靠到达的数据，就可以使用UDP协议。

服务器端流程如下：

1.创建socket对象，用SOCK_DGRAM指定Socket的类型是UDP

2.设置socket选项，当socket关闭后，本地端用于该socket的端口号立刻就可以被重用。

3.绑定端口及ip地址

4.接收数据 自动阻塞 等待客户端请求

5.发送数据给客户端

recvfrom()方法可以返回数据和客户端的地址与端口，这样，服务器收到数据后，直接调用sendto()就可以把数据用UDP发给客户端。

4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
[root@localhost day08]# vim udp_time_serv.py
#! /usr/bin/env python3

import socket
import time

host = '0.0.0.0'
port = 21567
addr = (host, port)

# SOCK_DGRAM指定了这个Socket的类型是UDP
s = socket.socket(type=socket.SOCK_DGRAM)
#设置socket选项，当socket关闭后，本地端用于该socket的端口号立刻就可以被重用。
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
#绑定 客户端口和地址:
s.bind(addr)

while True:
#接收数据 自动阻塞 等待客户端请求:
    data, cli_addr = s.recvfrom(1024)
    data = str(data, encoding='utf8')
    if data.strip() == '':
        break
    data = '[%s] %s' %(time.strftime('%H:%M:%d'),data)
    print(data)
    s.sendto(bytes(data,encoding='utf8'), cli_addr)
```

Top

```
27.     s.close()
```

步骤二：测试脚本执行

```
01.     [root@localhost day08]# python3 udp_time_serv.py
02.     [15:41:25] nihao
03.     [15:41:25] how are you?
04.     [15:42:25] I'm fine,thank you
05.     [15:50:25] hello
```

5 案例5：创建UDP时间戳客户端

5.1 问题

创建udp_time_client.py脚本，编写一个UDP客户端：

1. 连接服务器的21567
2. 接收用户从键盘上的输入
3. 发送接收到的字符串给服务器
4. 如果用户按ctrl + c则退出程序

5.2 方案

客户端使用UDP时，首先仍然创建基于UDP的Socket，然后，不需要调用connect()，直接通过sendto()给服务器发数据，通过recvfrom()接收数据

客户端流程如下：

1.创建socket对象，用SOCK_DGRAM指定Socket的类型是UDP

2.发送数据给服务器端口及ip地址

3.接收服务器端数据

5.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

```
01.     [root@localhost day08]# vim udp_time_client.py
02.     #! /usr/bin/env python3
03.
04.     import socket
05.
06.     host = '127.0.0.1'
07.     port = 12345
08.     addr = (host, port)
```

Top

```
#创建套接字
c = socket.socket( type=socket.SOCK_DGRAM)
data=input( ">: ")
#发送数据
c.sendto( bytes( data,"utf-8") , addr)
#接收数据
print( c.recvfrom( 1024) )
c.close( )
```

步骤二：测试脚本执行

```
[root@localhost day08]# python3 udp_time_client.py
>: nihao
(b'[15:41:25] nihao', ('127.0.0.1', 12345))
[root@localhost day08]# python3 udp_time_client.py
>: how are you?
(b'[15:41:25] how are you?', ('127.0.0.1', 12345))
[root@localhost day08]# python3 udp_time_client.py
>: I'm fine,thank you
(b"[15:42:25] I'm fine,thank you", ('127.0.0.1', 12345))
[root@localhost day08]# python3 udp_time_client.py
>: hello
(b'[15:50:25] hello', ('127.0.0.1', 12345))
```

Top