2-网络编程

2-1 TCP/IP协议相关概念

1.IP地址

IP 地址就是标识网络中设备的一个地址

作用：标识网络中的唯一一台设备。

2.端口与端口号

端口是传输数据的通道。通过IP地址找到对应设备，通过端口号找到对应的端口，然后通过端口把数据传输给应用程序。端口号可以标识唯一一个端口。

端口号分为知名端口号和动态端口号。

知名端口号指固定分配给一些服务的端口号，范围从0到1023。（比如21端口分配给FTP，25端口分配给SMTP，80端口分配给HTTP）

动态端口号指开发应用程序使用的端口号，范围从1024到65536，程序退出后端口号被释放。

3.TCP

TCP（Transmission Control Protocol）简称传输控制协议，它是一种面向连接的、可靠的、基于字节流的传输层通讯协议。

通信步骤：创建连接、传输数据、关闭连接

特点：

1、面向连接：通信双方必须先建立好连接才能进行数据传输，传输完成后断开连接释放系统资源

2、可靠传输：发送应答机制，超时重传、错误校验、流量控制和阻塞管理。

4.Socket

Socket（简称套接字）是进程之间通信的一个工具，负责进程之间的网络数据传输。

2-2 TCP客户端程序开发

1、开发步骤

创建客户端套接字对象--和服务器套接字建立连接--发送数据--接收数据--关闭客户端套接字。

2、Socket类

创建socket对象：socket.socket(AddressFamily,Type)

AddressFamily：表示IP地址类型，分为IPv4、IPv6

Type：表示传输协议类型

方法说明：

connect((host,port))表示和服务端套接字建立连接，host是服务器IP地址，port是应用程序端口号。

send(data)表示发送数据，data是二进制数据

recv(buffersize)表示接收数据，buffersize是每次接收的长度。

3、程序开发

```python
import socket

if __name__ == '__main__':
    # 创建tcp客户端套接字
    # 1. AF_INET: 表示ipv4
    # 2. SOCK_STREAM: tcp传输协议
    tcp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 和服务端应用程序建立连接
    tcp_client_socket.connect(("192.168.174.102", 8989))
     # 代码执行到此，说明连接建立成功
     # 准备发送的数据
     send_data = "你好服务端，我是客户端小黑!".encode("gbk")
     # 发送数据
     tcp_client_socket.send(send_data)
     # 接收数据，这次接收的数据最大字节数是1024
```

```
    recv_data = tcp_client_socket.recv(1024)
    # 返回的直接是服务端程序发送的二进制数据
    print(recv_data)
    # 对数据进行解码
    recv_content = recv_data.decode("gbk")
    print("接收服务端的数据为:", recv_content)
    # 关闭套接字
    tcp_client_socket.close()
```

2-3 TCP服务器端程序开发

1、开发步骤

创建服务器端的套接字对象--绑定端口号--设置监听--等待接受客户端的连接请求--接收数据--发送数据--关闭套接字

2、Socket类的相关方法说明

- bind((host, port)) 表示绑定端口号, host 是 ip 地址，port 是端口号，ip 地址一般不指定，表示本机的任何一个ip地址都可以。
- listen (backlog) 表示设置监听，backlog参数表示最大等待建立连接的个数。
- accept() 表示等待接受客户端的连接请求
- send(data) 表示发送数据，data 是二进制数据
- recv(buffersize) 表示接收数据, buffersize 是每次接收数据的长度

3、程序开发

```
import socket

if __name__ == '__main__':
    # 创建tcp服务端套接字
    tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 设置端口号复用，让程序退出端口号立即释放
    tcp_server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # 给程序绑定端口号
    tcp_server_socket.bind(("", 8989))
    # 设置监听
    # 128:最大等待建立连接的个数， 提示： 目前是单任务的服务端，同一时刻只能服务与一个客户端，后续使用多任务能够让服
    # 不需要让客户端进行等待建立连接
    # listen后的这个套接字只负责接收客户端连接请求，不能收发消息，收发消息使用返回的这个新套接字来完成
    tcp_server_socket.listen(128)
    # 等待客户端建立连接的请求，只有客户端和服务端建立连接成功代码才会解阻塞，代码才能继续往下执行
    # 1. 专门和客户端通信的套接字： service_client_socket
    # 2. 客户端的ip地址和端口号： ip_port
    service_client_socket, ip_port = tcp_server_socket.accept()
    # 代码执行到此说明连接建立成功
    print("客户端的ip地址和端口号:", ip_port)
    # 接收客户端发送的数据，这次接收数据的最大字节数是1024
    recv_data = service_client_socket.recv(1024)
    # 获取数据的长度
    recv_data_length = len(recv_data)
    print("接收数据的长度为:", recv_data_length)
    # 对二进制数据进行解码
    recv_content = recv_data.decode("gbk")
    print("接收客户端的数据为:", recv_content)
    # 准备发送的数据
    send_data = "ok，问题正在处理中...".encode("gbk")
    # 发送数据给客户端
```

```
    service_client_socket.send(send_data)
    # 关闭服务与客户端的套接字， 终止和客户端通信的服务
    service_client_socket.close()
    # 关闭服务端的套接字，终止和客户端提供建立连接请求的服务
    tcp_server_socket.close()
```

2-4 TCP程序开发的注意点

1、当 TCP 客户端程序想要和 TCP 服务端程序进行通信的时候必须要先**建立连接。**

2、TCP 客户端程序一般不需要绑定端口号，因为客户端是主动发起建立连接的。

3、TCP 服务端程序必须绑定端口号，否则客户端找不到这个 TCP 服务端程序。

4、listen 后的套接字是被动套接字，**只负责接收新的客户端的连接请求，不能收发消息。**

5、当 TCP 客户端程序和 TCP 服务端程序连接成功后， TCP 服务器端程序会产生一个**新的套接字**，收发客户端消息使用该套接字。

6、关闭 accept 返回的套接字意味着和这个客户端已经通信完毕。

7、关闭 listen 后的套接字意味着服务端的套接字关闭了，会导致新的客户端不能连接服务端，但是之前已经接成功的客户端还能正常通信。

8、当客户端的套接字调用 close 后，服务器端的 recv 会解阻塞，返回的数据长度为0，服务端可以通过返回数据的长度来判断客户端是否已经下线，反之**服务端关闭套接字，客户端的 recv 也会解阻塞，返回的数据长度也为0。**

2-5 案例： 多任务版TCP服务器端程序开发

```
import socket
import threading

def NLU(context):
    #做智能信息判断处理

def client_request(service_client_socket,ip_port):
    while True:
        recv_data = service_client_socket.recv(1024)
        if recv_data:
            context = recv_data.decode("gbk")
            print("接收到的消息是: ",recv_data.decode("gbk"))
            send_context = NLU(context)
            service_client_socket.send(send_context.encode("gbk"))
        else:
            print("客户端已下线",ip_port)
            break
    service_client_socket.close()

if __name__ == '__main__':
    tcp_server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    tcp_server_socket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,True)
    tcp_server_socket.bind(("",9090))
    tcp_server_socket.listen(128)
    while True:
        service_client_socket,ip_port=tcp_server_socket.accept()
        print("客户端连接成功: ",ip_port)
        sub_threading = threading.Thread(target=client_request,args=(service_client_socket,ip_port))
        sub_threading.setDaemon(True)
        sub_threading.start()
```

2-6 Socket中对send和recv原理剖析

1、Socket的缓冲区

Socket创建时会有一个发送缓冲区和一个接受缓冲区，指内存中一片空间。

2、send原理剖析

应用程序把发送的数据先写读到发送缓冲区，再由操作系统控制网卡把发送缓冲区的数据发送给服务端网卡。

3、recv原理剖析

由操作系统通过网卡接收数据，把接收的数据写入到接收缓冲区，应用程序从接收缓冲区获取客户端发送的数据。