python3编码转换

1. 二进制与字符串相互转换

> 编码：字符串类型的是数据转换为二进制数据
>
> 解码: 二进制数据还原为字符串数据

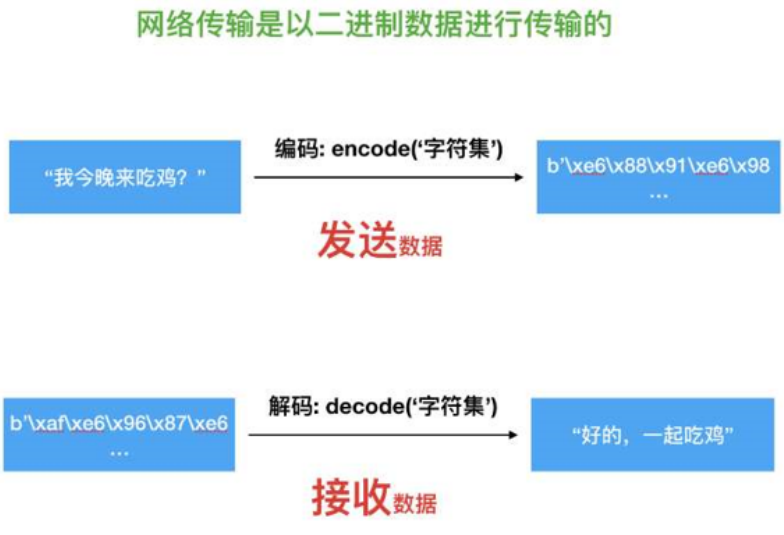
网络数据的传输:

- 网络传输是以二进制数据进行传输的

提示:

- 在网络传输数据的时候，数据需要先编码转化为二进制（bytes）数据类型

数据的编码转化:



函数名	说明
encode	编码 将字符串转化为字节码
decode	解码 将字节码转化为字符串

提示：

encoed()和decode()函数可以接受参数,encoding是指在编解码过程中使用的编码方案。

bytes.decode(encoding= “utf-8”)

str.encode(encoding=” utf-8”)

知识要点:

1. 网络传输是以二进制数据进行传输的

2. 数据转化用到了encode和decode函数

```
bytes.decode(encoding=“utf-8”)
str.encode(encoding=”utf-8”)
```

TCP客户端程序开发流程

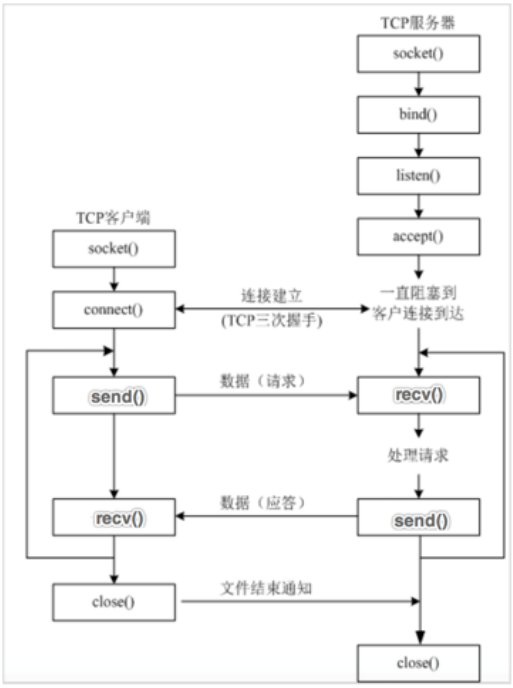
1.TCP网络应用程序开发分为:

```
TCP客户端程序开发：TCP Client
TCP服务端程序开发: TCP Server
```

提示:

```
客户端程序是指运行在用户设备上的程序
服务端程序是指运行在服务器设备上的程序，专门为客户端提供数据服务。
```

2.TCP客户端程序开发流程介绍



1.创建客户端套接字对象(买电话)

2.和服务端套接字建立连接(打电话)

3.发送数据(说话)

4.接收数据(接听)

5.关闭客户端套接字(挂电话)

知识要点

1.TCP网络应用程序开发分为客户端程序开发和服务端程序开发。

2. 主动发起建立连接请求的是客户端程序

3. 等待接受连接请求的是服务端程序

面试要点

> https://blog.csdn.net/m0_49330686/article/details/129400464

简述tcp三次握手，四次挥手过程

三次握手：

第一次握手：客户端主动发送SYN包到服务器，并进入SYN_SEND状态，等待服务器确认
第二次握手：服务器收到SYN包并确认，发送SYN+ACK到客户端，服务器进入SYN_RECV状态
第三次握手：客户端收到SYN+ACK包，发送ACK确认连接，发送完毕后客户端和服务端进入ESTABLISHED状态，完成三次握手

1. 客户端向服务器发生连接请求，并且等待服务器的确认
2. 服务器收到请求，确认链接，并且向客户端发送确认
3. 客户端收到了确认，发送最终的确认，连接建立完成

四次挥手：
第1次挥手：客户端发送一个FIN，用来关闭客户端到服务端的数据传送，客户端进入FIN_WAIT_1状态；
第2次挥手：服务端收到FIN后，发送一个ACK给客户端，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），服务端进入CLOSE_WAIT状态；
第3次挥手：服务端发送一个FIN，用来关闭服务端到客户端的数据传送，服务端进入LAST_ACK状态；
第4次挥手：客户端收到FIN后，客户端t进入TIME_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，服务端进入CLOSED状态，完成四次挥手

1. 客户端发送连接关闭请求
2. 服务器收到确认请求，并且停止发送数据
3. 服务器发送连接关闭
4. 客户端确认请求，连接关闭完成

TCP客户端程序开发

1.开发TCP客户端程序流程

1.创建客户端套接字对象
2.和服务端套接字建立连接
3.发送数据
4.接收数据
5.关闭客户端套接字

2.socket类的介绍

导入socket模块

```
import socket
```

创建客户端socket对象使用socket类

```
socket.socket(AddressFamily, Type)
```

客户端socket类的参数和方法说明:

参数名	说明
AddressFamily	IP地址类型, 分为TPv4和IPv6
Type	传输协议类型

3.开发客户端使用到的函数

方法名	说明
connect	和服务端套接字建立连接
send	发送数据
recv	接收数据
close	关闭连接

4.创建TCP客户端套接字

```
import socket
# 创建udp的套接字
# AF_INET: ipv4地址类型
# SOCK_STREAM: TCP传输协议类型
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
# ...这里是使用套接字的功能（省略）...
# 不用的时候，关闭套接字
s.close()
```

5.TCP客户端程序发送消息

```
import socket
# 创建tcp socket
tcp_client_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
# 目的信息
server_ip = input(“请输入服务器ip:”)
server_port = int(input(“请输入服务器port:”))
# 连接服务器套接字
tcp_client_socket.connect((server_ip,
server_port))
# 提示用户输入数据
send_data = input(“请输入要发送的数据：”)
tcp_client_socket.send(send_data.encode(”ut
f-8“)) # 注意：如果是windows的网络调试助手使用
gbk编码
# 关闭套接字
tcp_client_socket.close()
```

6.TCP客户端程序接收消息

```
import socket
# 创建tcp socket
tcp_client_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
server_ip = input("请输入服务器ip:")
server_port = int(input("请输入服务器port:"))
# 连接服务器套接字
```

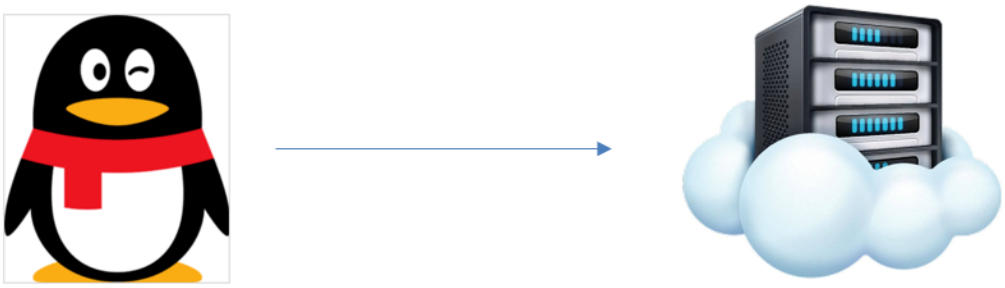
```
tcp_client_socket.connect((server_ip,
server_port))
send_data = input("请输入要发送的数据：")
tcp_client_socket.send(send_data.encode("ut
f-8"))
# 接收对方发送过来的数据，最大接收1024个字节
recvData = tcp_client_socket.recv(1024)
print('接收到的数据为:',
recvData.decode('utf-8'))
# 关闭套接字
tcp_client_socket.close()
```

7.知识要点

1. 导入socket模块
2. 创建TCP套接字‘socket’
   参数1：‘AF_INET’，表示IPv4地址类型
   参数2：‘SOCK_STREAM’，表示TCP传输协议类型
3. 发送数据‘send’
   参数1：要发送的二进制数据， 注意：字符串需要使用encode()方法进行编码
4. 接收数据‘recv’
   参数1：表示每次接收数据的大小，单位是字节
5. 关闭套接字‘socket’表示通信完成

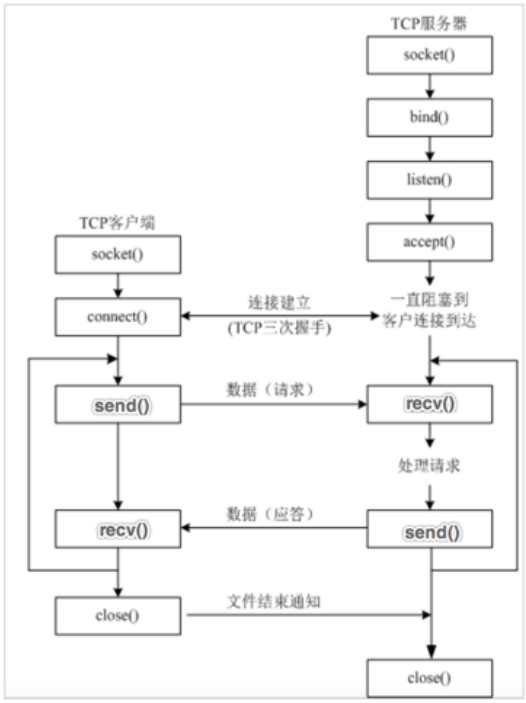
TCP服务端程序开发流程

服务端



具有了客户端和服务端，一个网络应用才可以真正的使用

1.TCP服务端程序开发流程介绍



1.创建服务端端套接字对象

2.绑定IP地址和端口号

3.设置监听

4.等待接受客户端的连接请求

5.接收数据

6.发送数据

7.关闭套接字

TCP服务端程序开发

1.socket类的介绍

导入socket模块

```
import socket
```

创建服务端socket对象使用socket类

```
socket.socket(AddressFamily, Type)
```

服务端socket类的参数和方法说明:

参数名	说明
AddressFamily	IP地址类型, 分为TPv4和IPv6
Type	传输协议类型

TCP服务端程序开发相关函数

方法名	说明
bind	绑定IP地址和端口号
listen	设置监听
accept	等待接受客户端的连接请求
send	发送数据
recv	接收数据

2.TCP服务端程序接收消息

```
import socket
# 创建socket
```

```
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_SOC
KET, socket.SO_REUSEADDR, True)
tcp_server_socket.bind(('', 7788)) #绑定IP地
址和端口号
# 设置监听
tcp_server_socket.listen(128) # 128:表示最大
等待连接数
# 如果有新的客户端来连接服务端，那么就产生一个新的套
接字专门为这个客户端服务
client_socket, clientAddr =
tcp_server_socket.accept()
recv_data = client_socket.recv(1024)  # 接收
1024个字节
print('接收到的数据为:',
recv_data.decode('utf-8'))
client_socket.close() # 关闭为这个服务与客户端
的套接字
tcp_server_socket.close() # 关闭为这个服务端套
接字
```

3.TCP服务端程序发送消息

```
import socket
# 创建socket
```

```
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_SOC
KET, socket.SO_REUSEADDR, True)
tcp_server_socket.bind(('', 7788)) #绑定IP地
址和端口号
tcp_server_socket.listen(128) # 设置监听，
128:表示最大等待连接数
# 如果有新的客户端来连接服务端，那么就产生一个新的套
接字专门为这个客户端服务
client_socket, clientAddr =
tcp_server_socket.accept()
recv_data = client_socket.recv(1024)  # 接收
1024个字节
print('接收到的数据为:',
recv_data.decode('utf-8'))
client_socket.send("thank you
!".encode('utf-8'))  # 发送数据到客户端
client_socket.close() # 关闭为这个服务与客户端
的套接字
tcp_server_socket.close() # 关闭为这个服务端套
接字
```

知识要点

1. 导入socket模块
2. 创建TCP套接字'socket'
   参数1: 'AF_INET', 表示IPv4地址类型

参数2：‘SOCK_STREAM’，表示TCP传输协议类型

3．绑定端口号‘bind’

参数1：　元组，比如:(’’，　端口号)，元组里面的一个元素是ip地址，一般不需要设置，第二个元素是启动程序后使用的端口号。

4．设置监听‘listen’

参数1：　最大等待连接数

5．等待接受客户端的连接请求‘accept’

6．发送数据‘send’

参数1：要发送的二进制数据，　注意：字符串需要使用encode()方法进行编码

7．　接收数据‘recv’

参数1：表示每次接收数据的大小，单位是字节，注意：解码成字符串使用decode()方法

8．　关闭套接字‘socket’表示通信完成

TCP网络应用程序注意点的介绍

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. 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。

@socket之send和recv原理剖析

@TCP socket的发送和接收缓冲区

当创建一个TCP socket对象的时候会有一个发送缓冲区和一个接收缓冲区，这个发送和接收缓冲区指的就是内存中的一片空间。

@send原理剖析

send是不是直接把数据发给服务端?

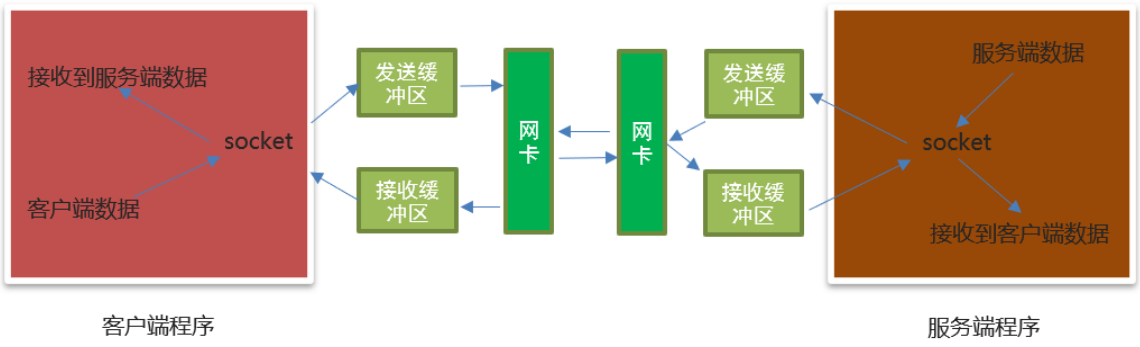
不是，要想发数据，必须得通过网卡发送数据，应用程序是无法直接通过网卡发送数据的，它需要调用操作系统接口，也就是说，应用程序把发送的数据先写入到发送缓冲区(内存中的一片空间)，再由操作系统控制网卡把发送缓冲区的数据发送给服务端网卡

@recv原理剖析

recv是不是直接从客户端接收数据?

不是，应用软件是无法直接通过网卡接收数据的，它需要调用操作系统接口，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(内存中的一片空间），应用程序再从接收缓存区获取客户端发送的数据。

@send和recv原理剖析图



说明:

发送数据是发送到发送缓冲区，接收数据是从接收缓冲区

@知识要点

不管是recv还是send都不是直接接收到对方的数据和发送数据到对方，发送数据会写入到发送缓冲区，接收数据是从接收缓冲区来读取，发送数据和接收数据最终是由操作系统控制网卡来完成。

@案例-多任务版TCP服务端程序开发

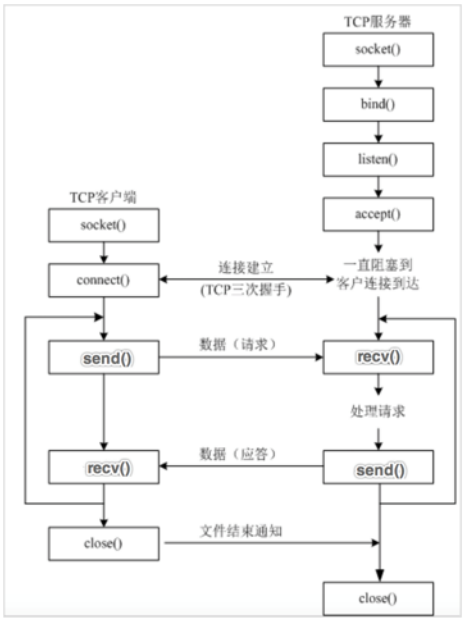
思考：？

目前我们开发的TCP服务端程序只能服务于一个客户端

如何实现一个服务端服务多个客户端？

实现步骤分析:

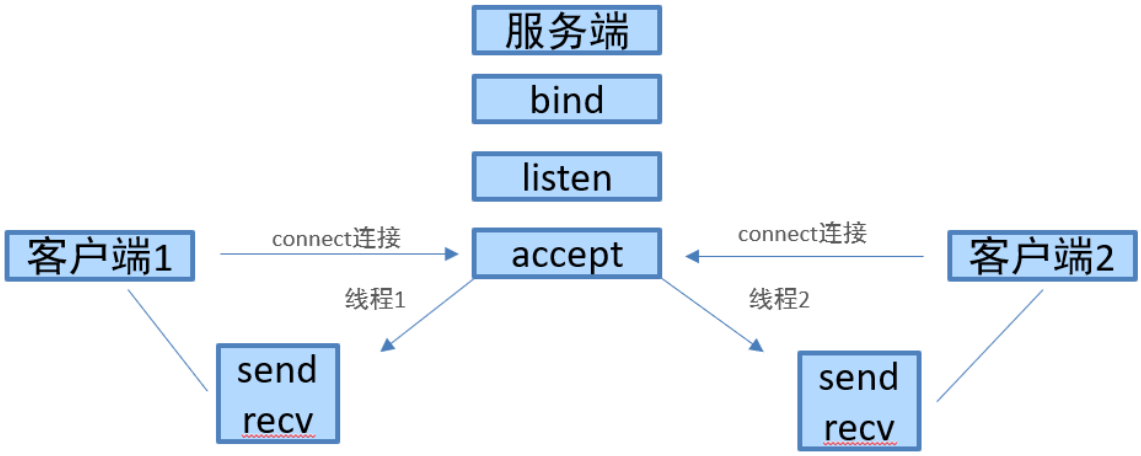
1. 编写一个TCP服务端程序，循环等待接受客户端的连接请求



目前我们开发的TCP服务端程序不能同时服务于多个客户端使用多任务可以实现一个服务端同时服务多个客户端，本案例中我们使用线程

实现步骤分析:

当客户端和服务端建立连接成功，创建子线程，使用子线程专门处理客户端的请求，防止主线程阻塞



示例代码:

```
tcp_server_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM) # 创建tcp服务端套接字
# 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_SOC
KET, socket.SO_REUSEADDR, True)
tcp_server_socket.bind((“”, 9090))  # 绑定端
口号
# 设置监听，listen后的套接字是被动套接字，只负责接
收客户端的连接请求
tcp_server_socket.listen(128)
# 循环等待接收客户端的连接请求
```

```
while True:
    service_client_socket, ip_port =
tcp_server_socket.accept() # 等待接收客户端的
连接请求
    print("客户端连接成功:", ip_port)
    # 当客户端和服务端建立连接成功以后，需要创建一
个子线程，不同子线程负责接收不同客户端的消息
    sub_thread =
threading.Thread(target=handle_client_reque
st, args=(service_client_socket, ip_port))
    sub_thread.setDaemon(True) # 设置守护主线
程
    sub_thread.start() # 启动子线程
```

@知识要点

> 20:51上课

1. 编写一个TCP服务端程序，循环等待接受客户端的连接请求

```
    while True:
        service_client_socket, ip_port = tcp_server_socket.accept()
```

2. 当客户端和服务端建立连接成功，创建子线程，使用子线程专门处理客户端的请求，防止主线程阻塞

```
    while True:
        service_client_socket, ip_port = tcp_server_socket.accept()
        sub_thread = threading.Thread(target=handle_client_request, args=(service_client_socket, ip_port))
        sub_thread.start()
```