3.1 今日目标

3.2 [重点]`IP`地址的介绍

- IP地址的表现形式:
  - `IPv4`: 点分十进制
  - `IPv6`: 冒号十六进制
- IP地址的作用:标识网络中唯一的一台设备的
- 查看IP地址:
  - `Linux`和`mac OS`:`ifconfig`
  - `Windows`:`ipconfig`
- 检查网络是否正常:
  - `ping www.baidu.com`
  - `ping 192.168.160.`
  - `ping 127.0.0.1`本地回坏地址

3.3 [重点]端口和端口号的介绍

- 端口：传输数据的通道(教室门)
- 端口号: 端口的唯一标识(门牌号)
- 关系: 端口号可以标识唯一的一个端口
- 端口号分类：
  - 知名端口号的范围是0到1023
  - 动态端口号的范围是1024到65535

3.4 [重点]TCP介绍

- 概念：**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信协议。**
- TCP 通信步骤：
  - 创建连接
  - 传输数据
  - 关闭连接
- TCP 的特点：
  - 面向连接
  - 可靠传输: 保证数据不丢包\不乱序

3.5 [难点]socket介绍

- 概念：socket (简称 套接字) 是**进程之间通信一个工具**，好比现实生活中的**插座**
- 作用：**进程之间的网络数据传输**
- 使用场景：**网络相关的应用程序或者软件都使用到了 socket**

3.6 [重点]TCP网络应用程序开发流程

- 客户端开发的流程：主动连接的

```
1. 创建套接字对象
2. 连接服务端
3. 发送数据
4. 接受数据
5. 关闭连接
```

- 服务端开发的流程：等待连接的

```
1. 创建套接字对象
2. 绑定端口
3. 开始监听
4. 等待连接
5. 发送数据
6. 接受数据
7. 关闭连接
```

3.7 [重点]TCP客户端程序开发

- 客户端 开发

```
"""
0. 导入模块
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
"""

# 0. 导入模块
import socket


if __name__ == '__main__':


    """
    Windows: 网络助手，接受消息类型UTF-8，发送消息类型GBK
    Ubuntu: 网络助手，接受消息类型UTF-8，发送消息类型UTF-8
    """

    # 1. 创建客户端套接字对象
    # AddressFamily 表示IP地址类型，分为IPv4和IPv6   AF_INET: IPv4
    # Type 表示传输类型: 流式套接字类型-->默认就是TCP
    # Proto  传输协议: 默认就是0--> 当选择了流失套接字，协议默认是0，就是TCP协议
    tcp_client_socket = socket.socket()
    # tcp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)

    # 2. 和服务端套接字建立连接
```

```
    # connect(元组)
    # 元组(host, port)
    # 因为服务端监听8080端口器，所以客户端连接时要填入8080
    tcp_client_socket.connect(("192.168.160.130", 8080))

    # 3. 发送数据
    # send(二进制数据)
    # 数据.encode()
    # 编码: UTF-8 \ GBK
    send_data = "你好，服务端".encode("UTF-8")
    tcp_client_socket.send(send_data)

    # 4. 接收数据
    # recv(每次接受的字节长度)
    rec_data = tcp_client_socket.recv(1024)
    print(rec_data)

    # 接受的数据进行解码
    print(rec_data.decode("UTF-8"))

    # 5. 关闭客户端套接字
    tcp_client_socket.close()
```

3.8 [难点]TCP服务端程序开发

- 服务端 开发

```
"""
0. 导入模块
1. 创建服务端端套接字对象
2. 绑定端口号
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
"""

# 0. 导入模块
import socket

if __name__ == '__main__':
    # 1. 创建服务端端套接字对象
    # AddressFamily 表示IP地址类型，分为IPv4和IPv6   AF_INET: IPv4
    # Type 表示传输类型: 流式套接字类型-->默认就是TCP
    # Proto  传输协议: 默认就是0--> 当选择了流失套接字，协议默认是0，就是TCP协议
    tcp_server_socket = socket.socket()

    # 2. 绑定端口号
    # bing(元组)
    # 元组(host, port)
    # IP不写，默认是全零地址，表示可以接收任何一个能访问的IP: 局域网\公网\本地回环地址
    tcp_server_socket.bind(("", 8080))
```

```python
# 3. 设置监听
# bakclog: 设置等待的最大连接数. 建议128, 具体多少个, 系统决定
# 设置了listen之后, 就意味着这个socket只能监听, 不能收发消息
tcp_server_socket.listen(128)

# 4. 等待接受客户端的连接请求
# 1. 专门和客户端通信的套接字:  service_client_socket(生成了一个新的socket)
# 2. 客户端的ip地址和端口号:  ip_port
new_client_socket, ip_port = tcp_server_socket.accept()
print(new_client_socket, ip_port)

# 5. 接收数据
# recv(每次接受的字节长度)
recv_data = new_client_socket.recv(1024)
# 数据解码
print("接收到客户端消息:", recv_data.decode("UTF-8"))

# 6. 发送数据
send_data = "我是服务器," + "你好客户端:" + str(ip_port)
new_client_socket.send(send_data.encode("UTF-8"))

# 7. 关闭套接字
# 关闭服务与客户端的套接字,  终止和客户端通信的服务
new_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()
```

3.9 [重点]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。(会自动分配)
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。

3.10 [重点]案例-多任务版TCP服务端程序开发

- 需求:

  开发一个多任务版的TCP服务端程序能够服务于多个客户端

- 具体实现步骤:

```
"""
0. 导入模块
1. 创建服务端端套接字对象
2. 绑定端口号
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
"""

# 0. 导入模块
import socket
import threading


# 处理客户端的请求操作
def handle_client_request(service_client_socket, ip_port):
    # 循环接收客户端发送的数据
    while True:
        # 接收客户端发送的数据
        recv_data = service_client_socket.recv(1024)
        # 容器类型判断是否有数据可以直接使用if语句进行判断，如果容器类型里面有数据表示条件成立，否则条件失败
        # 容器类型: 列表、字典、元组、字符串、set、range、二进制数据
        if recv_data:
            print(recv_data.decode("UTF-8"), ip_port)
            # 回复
            service_client_socket.send("ok, 问题正在处理中...".encode("UTF-8"))

        else:
            print("客户端下线了:", ip_port)
            break
    # 终止和客户端进行通信
    service_client_socket.close()


if __name__ == '__main__':
    # 1. 创建服务端端套接字对象
    tcp_server_socket = socket.socket()

    # 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)

    # 2. 绑定端口号
    tcp_server_socket.bind(("", 8080))

    # 3. 设置监听
    tcp_server_socket.listen(128)

    # 4. 等待接受客户端的连接请求
    while True:
        # 循环接收新的客户端
```

```
        new_client_socket, ip_port = tcp_server_socket.accept()
        print(new_client_socket, ip_port)

        # 实现子线程
        client_socket_threading = threading.Thread(target=handle_client_request, args=
(new_client_socket, ip_port))
        # 设置守护主线程
        client_socket_threading.setDaemon(True)
        # 启动子线程
        client_socket_threading.start()


    # 服务端一般不用关闭
    # tcp_server_socket.close()
```

3.11 [难点]socket之send和recv原理剖析

- Send 原理

```
发消息send()-->发送缓冲区-->网卡-->网卡-->接收缓冲区-->接收消息recv()
```

- Recv 原理

```
接收消息recv()<--接收缓冲区<--网卡<--网卡<--发送缓冲区<--发消息send()
```

3.12 今日知识总结