# 课程介绍

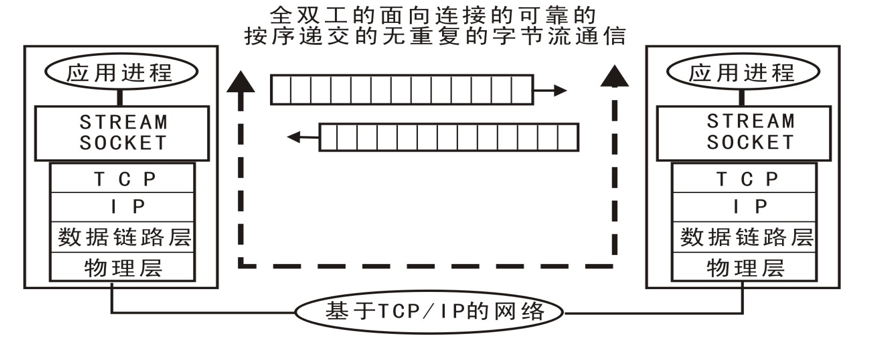
* 3. TCP服务、客户编程;（掌握）
* 4. TCP的状态；（了解）
* 5. TCP和UDP通信总结；（了解）

1. TCP服务器和客户端
   1. 什么是TCP协议

TCP协议，传输控制协议（Transmission Control Protocol，缩写为TCP）是一种**面向连接的**、可靠的、基于字节流的传输层通信协议，它能提供高可靠性通信(即数据无误、数据无丢失、数据无失序、数据无重复到达的通信)：

【使用场景】：

* 适合于对传输质量要求较高，以及传输大量数据的通信。
* 在需要可靠数据传输的场合，通常使用TCP协议。
* MSN/QQ等即时通讯软件的用户登录账户管理相关的功能通常采用TCP协议。



tcp通信需要经过**创建连接**、**数据传送**、**终止连接**三个步骤。

* + 1. 面向连接的

通信双方必须先建立连接才能进行数据的传输，双方都必须为该连接分配必要的系统内核资源，以管理连接的状态和连接上的传输。

完成数据交换后，双方必须断开此连接，以释放系统资源。

这种连接是一对一的，不适用于广播的应用程序，基于广播的应用程序适合使用UDP协议。

* + 1. 可靠的
* TCP采用发送应答机制

TCP发送的每个报文段都必须得到接收方的应答才认为这个TCP报文段传输成功

* 超时重传

发送端发出一个报文段之后就启动定时器，如果在定时时间内没有收到应答就重新发送这个报文段。

TCP为了保证不发生丢包，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的包发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据包就被假设为已丢失将会被进行重传。

* 错误校验

TCP用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算校验和。

* 流量控制和阻塞管理

流量控制用来避免主机发送得过快而使接收方来不及完全收下。

* 1. TCP服务端基本模型

可以想象成一个公司售后的人工服务台，要想为客户提供服务，我们需要怎么做那：

1、买个电话机；

2、插上电话卡，这个时候就有一个号码了；

3、请一个接听电话的工作人员A，然后告诉所有的客户，我们公司的售后服务电话号码；

4、当客户打入电话后，A把这个客户分配给一个专门的客服人员B1进行沟通，而A继续等待其他新的客户打入电话；

5、客服人员B1和这个客户进行电话交流；

如同上面的售后服务公司一样，TCP的服务器流程是一样的：

1、通过socket创建一个套接字对象。（**申请对象**）

2、这个套接字对象绑定ip和port。（**身份配置**）

3、通过listen使这个套接字变为可以被动连接的状态。（**身份调整**）

4、通过accept获取一个新的客户连接。（**开辟新连接**）

5、新的客户连接通过recv/send接收发送数据。（**维持新连接**）

from socket import \*

# 创建socket

tcpSerSocket = socket(AF\_INET, SOCK\_STREAM)

# 绑定本地信息

address = ('', 7788)

tcpSerSocket.bind(address)

# 使用socket创建的套接字默认的属性是主动的，使用listen将其变为被动的，这样就可以接收别人的链接了

tcpSerSocket.listen(5)

# 如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端服务器

# newSocket用来为这个客户端服务

# tcpSerSocket就可以省下来专门等待其他新客户端的链接

newSocket, clientAddr = tcpSerSocket.accept()

# 接收对方发送过来的数据，最大接收1024个字节

recvData = newSocket.recv(1024)

print '接收到的数据为:',recvData

# 发送一些数据到客户端

newSocket.send("thank you !")

# 关闭为这个客户端服务的套接字，只要关闭了，就意味着为不能再为这个客户端服务了，如果还需要服务，只能再次重新连接

newSocket.close()

# 关闭监听套接字，只要这个套接字关闭了，就意味着整个程序不能再接收任何新的客户端的连接

tcpSerSocket.close()

* 1. TCP客户端基本模型

TCP客户端的核心接口API是connect，由于TCP要求的是稳定传输，所以在后面数据传输前，需要借助三次握手的协议特点去实现连接。而connect函数则是三次握手协议的发起者。

from socket import \*

# 创建socket

tcpClientSocket = socket(AF\_INET, SOCK\_STREAM)

# 链接服务器

serAddr = ('192.168.1.102', 7788)

tcpClientSocket.connect(serAddr)

# 提示用户输入数据

sendData = raw\_input("请输入要发送的数据：")

tcpClientSocket.send(sendData)

# 接收对方发送过来的数据，最大接收1024个字节

recvData = tcpClientSocket.recv(1024)

print '接收到的数据为:',recvData

# 关闭套接字

tcpClientSocket.close()

* 1. TCP客户服务器编程模型

通过上面的程序分析，我们这里看一个简单的TCP的C/S模型通信图。

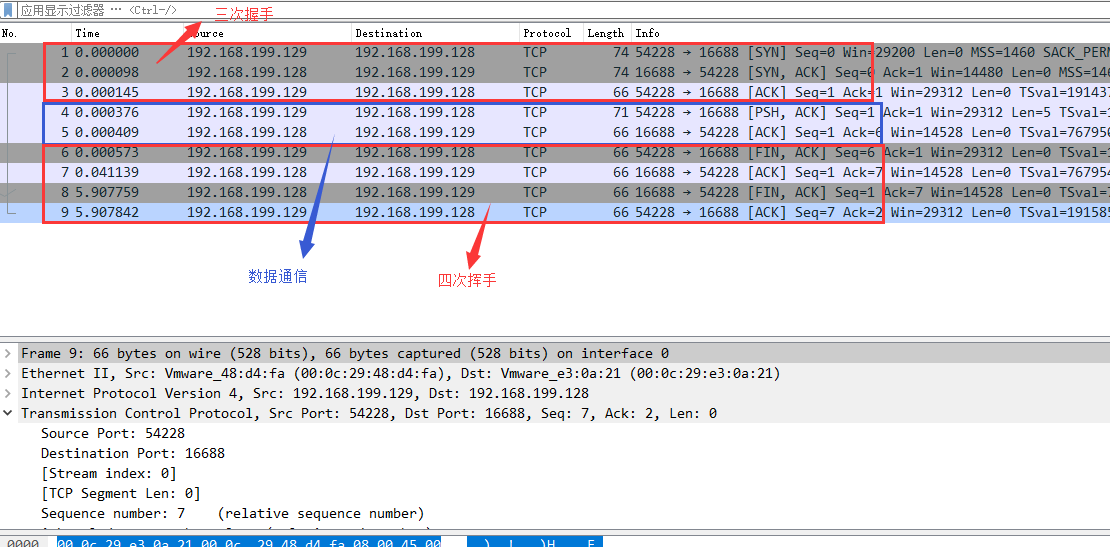


几个注意点：

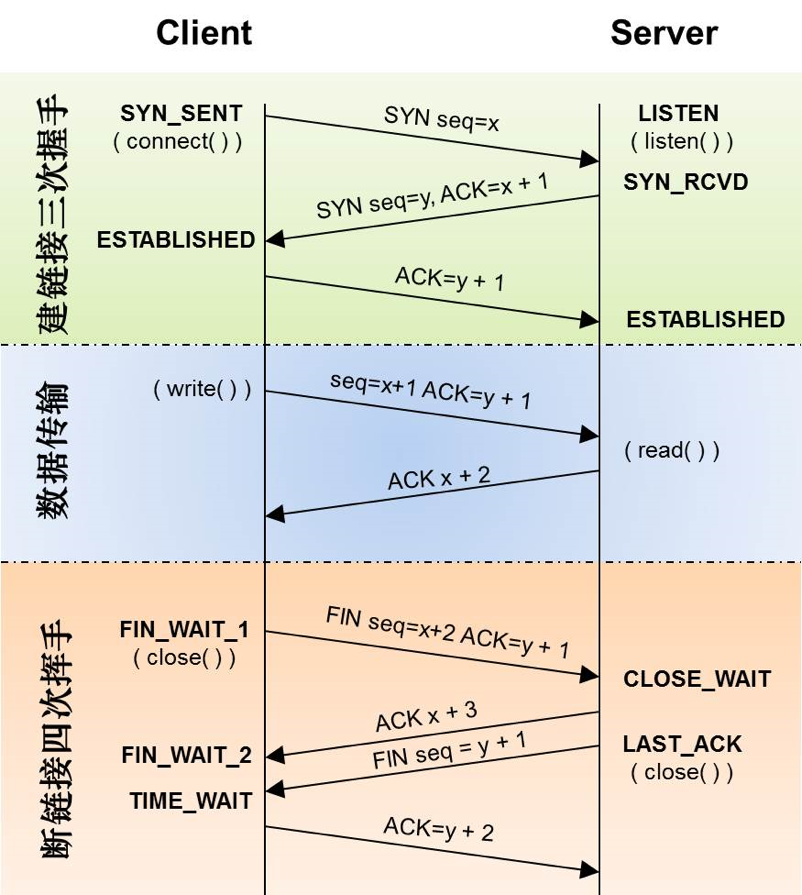
1、TCP的监听状态是在listen调用后就产生了；即使没有调用accept函数，也不影响客户端的connect；

2、connect服务器时，如果服务器没有，则会有超时异常的抛出。

1. TCP的三次握手和四次挥手（了解）
   1. TCP客户服务器通信抓包信息



* 1. TCP通信状态图



* 1. TCP的三次握手

tcp通过connect函数发起了第一次握手，这里一定要注意，三次握手的过程是在系统内核层上完成的。



* **第一次握手**：客户端首先将标志位SYN置为1，然后随机产生一个值seq=X的包，并将该数据包发送给server端，Client进入SYN\_SENT状态，等待server确认。
* **第二次握手**：server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack numbern=X+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。
* **第三次握手**：Client收到确认后，检查ack number是否为X+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。
  1. TCP的四次挥手
* **第一次分手**：主机1（可以使客户端，也可以是服务器端），设置Sequence Number和Acknowledgment Number，向主机2发送一个FIN报文段；此时，主机1进入FIN\_WAIT\_1状态；这表示主机1没有数据要发送给主机2了；
* **第二次分手**：主机2收到了主机1发送的FIN报文段，向主机1回一个ACK报文段，Acknowledgment Number为Sequence Number加1；主机1进入FIN\_WAIT\_2状态；主机2告诉主机1，我“同意”你的关闭请求；
* **第三次分手**：主机2向主机1发送FIN报文段，请求关闭连接，同时主机2进入LAST\_ACK状态；
* **第四次分手**：主机1收到主机2发送的FIN报文段，向主机2发送ACK报文段，然后主机1进入TIME\_WAIT状态；主机2收到主机1的ACK报文段以后，就关闭连接；此时，主机1等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机1也可以关闭连接了。
  + 1. CLOSE\_WAIT状态

这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以 close这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。

* + 1. 2MLS问题
* 2MSL即两倍的MSL，TCP的TIME\_WAIT状态也称为2MSL等待状态，
* 当TCP的一端发起主动关闭，在发出最后一个ACK包后，即发送了第四次挥手的ACK包后就进入了TIME\_WAIT状态，必须在此状态上停留两倍的MSL时间，
* 等待2MSL时间主要目的是怕最后一个ACK包对方没收到，那么对方在超时后将重发第三次挥手的FIN包，主动关闭端接到重发的FIN包后可以再发一个ACK应答包。
* 在TIME\_WAIT状态时两端的端口不能使用，要等到2MSL时间结束才可继续使用。
* 当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。
* 不过在实际应用中可以通过设置SO\_REUSEADDR选项达到不必等待2MSL时间结束再使用此端口。

1. 课程总结
   1. 重点
      * 1. TCP的客户、服务器编程模型及代码编写
        2. TCP的缓存概念理解
   2. 难点
      * 1. TCP的三次握手（知道）
        2. TCP的四次挥手（知道）
2. 课后练习
   * + 1. 编写TCP服务器，客户端的代码，最后只看注释，能够写出代码。
       2. 实现利用TCP协议的1V1聊天程序。
3. 面试题
   * + 1. 谈一谈你对TCP和UDP的认识

TCP和UDP都是基于传输层的协议，定义了端口的概念；

TCP是面向数据流的，经过发送确认的机制确保了数据的无丢失传输；

UDP是面向数据报的，设计简单方便，但不保证无丢失传输；

1. 扩展知识或课外阅读推荐（可选）
   1. 单播、广播、组播通信
      1. 单播

单播是指数据包在计算机网络的传输中，目的地址为单一目标的一种传输方式。

它是现今网络应用最为广泛，通常所使用的网络协议或服务大多采用单播传输，例如一切基于TCP的协议。

* + 1. 广播

广播是指将信息数据包发往指定网络范围内的所有设备。其发送范围称为“广播域”。 通常来说，广播都是限制在局域网范围内的。

广播域的范围：

* **受限地址**，受限的广播地址是255.255.255.255。该地址用于主机配置过程中IP数据包的目的地址，此时，主机可能还不知道它所在网络的网络掩码，甚至连它的IP地址也不知道。
* **指向网络**，指向网络的广播地址是主机号为全1的地址。

使用广播的常用协议有：ARP和DHCP。

在socket编程时，需要先进行setsockopt将默认socket的属性更改为可以广播的属性。

import socket

# 创建udp套接字

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

# 告诉系统内核刚创建的套接字用来进行广播

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

data = input("请输入要广播的内容：")

while data:

# 注意对于广播对象地址的设置

# <broadcast>表示广播地址

s.sendto(data.encode("utf-8"), ("<broadcast>", 8080))

data = input("请输入要广播的内容：")

s.close()

* + 1. 组播

组播，有时候也叫多播，是指把信息同时传递给一组目的地址。他和广播相比，不会造成局域网的广播风暴，由于是特定组的数据通信，他使用的是D类IP池做为组播地址。

比较常用的场景：

1、向多个目的地址传送数据；

2、客户对服务器的请求，比如无盘工作站；

* 1. TCP三次握手、四次挥手详解

TCP三次握手和四次挥手：<https://www.jianshu.com/p/9968b16b607e>