1. **OSI七层模型，简要概述。**

OSI 七层模型

应用层 ：提供用户服务，具体内容由特定程序规定

表示层 ：提供数据的压缩解压和加密等

会话层 ：建立程序级的连接，决定使用什么样的传输服务

传输层 ：提供合适的通信服务，流量控制

网络层 ： 路由选择，网络互联，分组传输

链路层 ： 提供链路交换，具体消息发送

物理层 ： 物理硬件，接口

OSI 模型优点 ： 将功能分开，降低网路传输中的耦合度，每一部分完成自己的功能，开发更加清晰。

1. **什么是‘三次握手，四次挥手’？**

三次握手

1.客户端向服务器发起连接请求（问是否可以连接）

2.服务器接收到请求后进行确认（允许连接）返回报文

3.客户端收到许可，建立连接

四次挥手

1.主动方发送报文告知被动方要断开连接

2.被动方返回报文，告知受到请求，准备断开

3.被动方发送报文给主动方告知准备就绪可以断开

4.主动方发送报文确定断开

1. **什么是ARP？**

地址解析协议，即ARP（Address Resolution Protocol），是根据[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80" \t "https://baike.baidu.com/item/ARP/_blank)获取[物理地址](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%9C%B0%E5%9D%80/2129" \t "https://baike.baidu.com/item/ARP/_blank)的一个[TCP/IP协议](https://baike.baidu.com/item/TCP/IP%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/ARP/_blank)。[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA/455151" \t "https://baike.baidu.com/item/ARP/_blank)发送信息时将包含目标IP地址的ARP请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。地址解析协议是建立在网络中各个主机互相信任的基础上的，网络上的主机可以自主发送ARP应答消息，其他主机收到应答报文时不会检测该报文的真实性就会将其记入本机ARP缓存；由此攻击者就可以向某一主机发送伪ARP应答报文，使其发送的信息无法到达预期的主机或到达错误的主机，这就构成了一个[ARP欺骗](https://baike.baidu.com/item/ARP%E6%AC%BA%E9%AA%97" \t "https://baike.baidu.com/item/ARP/_blank)。[ARP命令](https://baike.baidu.com/item/ARP%E5%91%BD%E4%BB%A4" \t "https://baike.baidu.com/item/ARP/_blank)可用于查询本机ARP缓存中IP地址和[MAC地址](https://baike.baidu.com/item/MAC%E5%9C%B0%E5%9D%80" \t "https://baike.baidu.com/item/ARP/_blank)的对应关系、添加或删除静态对应关系等。相关协议有[RARP](https://baike.baidu.com/item/RARP" \t "https://baike.baidu.com/item/ARP/_blank)、[代理ARP](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86ARP" \t "https://baike.baidu.com/item/ARP/_blank)。[NDP](https://baike.baidu.com/item/NDP" \t "https://baike.baidu.com/item/ARP/_blank)用于在[IPv6](https://baike.baidu.com/item/IPv6" \t "https://baike.baidu.com/item/ARP/_blank)中代替地址解析协议。

1. **TCP，UDP的区别是什么？**

TCP的优点： 可靠，稳定 TCP的可靠体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。 TCP的缺点： 慢，效率低，占用系统资源高，易被攻击 TCP在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的CPU、内存等硬件资源。 而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

UDP的优点： 快，比TCP稍安全 UDP没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以它在传递数据时非常快。没有TCP的这些机制，UDP较TCP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击…… UDP的缺点： 不可靠，不稳定 因为UDP没有TCP那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。 基于上面的优缺点，那么： 什么时候应该使用TCP： 当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如HTTP、HTTPS、FTP等传输文件的协议，POP、SMTP等邮件传输的协议。 在日常生活中，常见使用TCP协议的应用如下： 浏览器，用的HTTP FlashFXP，用的FTP Outlook，用的POP、SMTP Putty，用的Telnet、SSH QQ文件传输 ………… 什么时候应该使用UDP： 当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快，这时就可以使用UDP。 比如，日常生活中，常见使用UDP协议的应用如下： QQ语音 QQ视频 TFTP ……

TCP与UDP的区别：

1.基于连接与无连接；  
 2.对系统资源的要求（TCP较多，UDP少）；  
 3.UDP程序结构较简单；  
 4.流模式与数据报模式 ；

5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

1. **什么是广域网，什么是局域网？**

1、广域网，英语名字为Wide Area Network，缩写为 WAN，又称广域网、外网、公网。指的是连接不同地区局域网或城域网计算机通信的远程网。通常跨接很大的物理范围，所覆盖的范围从几十公里到几千公里，它能连接多个地区、城市和国家，或横跨几个洲并能提供远距离通信，形成国际性的远程网络。一般所指的互联网是属于一种公共型的广域网。

2、局域网，英文名字Local Area Network，缩写为LAN。是指在某一区域内由多台计算机互联成的计算机组。一般是方圆几千米以内。局域网是封闭型的，可以由办公室内的两台[计算机组成](https://www.baidu.com/s?wd=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)，也可以由一个公司内的上千台[计算机组成](https://www.baidu.com/s?wd=%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)。

3、广域网（WAN）和局域网（LAN）的区别：

(1)局域网（LAN）主要是指在小范围内的计算机互联网络，其范围要比广域网（WAN）小得多。

(2)广域网上的每一台电脑都有一个或多个广域网IP地址，广域网IP地址一般要到ISP处交费之后才能申请到，广域网IP地址不能重复；局域网上的每一台电脑都有一个或多个局域网IP地址，局域网IP地址是局域网内部分配的，不同局域网的IP地址可以重复，不会相互影响。

1. **详细解释socket**

Python 提供了两个基本的 socket 模块。

  第一个是 Socket，它提供了标准的 BSD Sockets API。

    第二个是 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

下面讲的是Socket模块功能

1、Socket 类型

套接字格式：

socket(family,type[,protocal]) 使用给定的地址族、套接字类型、协议编号（默认为0）来创建套接字。

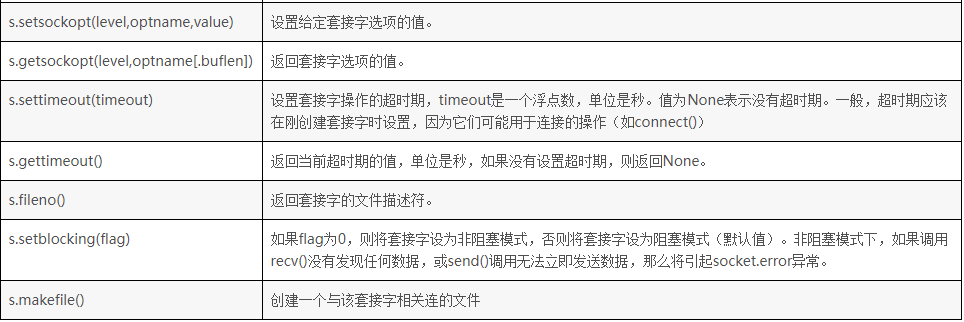


2、Socket 函数

1）TCP发送数据时，已建立好TCP连接，所以不需要指定地址。UDP是面向无连接的，每次发送要指定是发给谁。

2）服务端与客户端不能直接发送列表，元组，字典。需要字符串化repr(data)。





1. **TCP编程的服务器端一般步骤**

TCP服务端：

1 创建套接字，绑定套接字到本地IP与端口

    # socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) , s.bind()

2 开始监听连接                   #s.listen()

3 进入循环，不断接受客户端的连接请求              #s.accept()

4 然后接收传来的数据，并发送给对方数据         #s.recv() , s.sendall()

5 传输完毕后，关闭套接字                     #s.close()

1. **TCP编程的客户端一般步骤**

TCP客户端:

1 创建套接字，连接远端地址

        # socket.socket(socket.AF\_INET,socket.SOCK\_STREAM) , s.connect()

2 连接后发送数据和接收数据          # s.sendall(), s.recv()

3 传输完毕后，关闭套接字          #s.close()

1. **什么叫粘包？**

socket粘包：

socket 交互send时，连续处理多个send时会出现粘包，soket会把两条send作为一条send强制发送，会粘在一起。

send发送会根据recv定义的数值发送一个固定的数值，如果最后一次，所剩的数值小于recv定义数就会连带两条send数据同时发送，发生粘包状况。

解决方案：

方案1：可以使用time.sleep 在两send之间加上时间（不建议）

方案2：可以在send两条之间加入一条 conn.recv(1024)

服务端：  
conn.send(str(len(cmd\_res.encode())).encode(**"utf-8"**))   
client\_ack = conn.recv(1024) #wait client to confirm  
conn.send(cmd\_res.encode(**"utf-8"**))

客户端：  
client.send(**"准备好接收了,loser可以发了"**.encode(**"utf-8"**))

1. **什么是IO多路复用？IO多路复用适用于哪些场合？**

IO多路复用中包括 select、pool、epoll，这些都属于同步，还不属于异步

一、IO多路复用介绍

1、select

select最早于1983年出现在4.2BSD中，它通过一个select()系统调用来监视多个文件描述符的数组，当select()返回后，该数组中就绪的文件描述符便会被内核修改标志位，使得进程可以获得这些文件描述符从而进行后续的读写操作。

　　 select目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点， 事实上从现在看来，这也是它所剩不多的优点之一。

　　 select的一个缺点在于单个进程能够监视的文件描述符的数量存在最大限制， 在Linux上一般为1024，不过可以通过修改宏定义甚至重新编译内核的方式提升 这一限制。

　　 另外，select()所维护的存储大量文件描述符的数据结构，随着文件描述符数 量的增大，其复制的开销也线性增长。同时，由于网络响应时间的延迟使得大量 TCP连接处于非活跃状态，但调用select()会对所有socket进行一次线性扫描， 所以这也浪费了一定的开销。

2、poll

poll在1986年诞生于System V Release 3，它和select在本质上没有多大差别，但是poll没有最大文件描述符数量的限制。

　　 poll和select同样存在一个缺点就是，包含大量文件描述符的数组被整体复 制于用户态和内核的地址空间之间，而不论这些文件描述符是否就绪，它的开销随 着文件描述符数量的增加而线性增大。

　　 另外，select()和poll()将就绪的文件描述符告诉进程后，如果进程没有对其 进行IO操作，那么下次调用select()和poll()的时候将再次报告这些文件描述符， 所以它们一般不会丢失就绪的消息，这种方式称为水平触发（Level Triggered）。

3、epoll

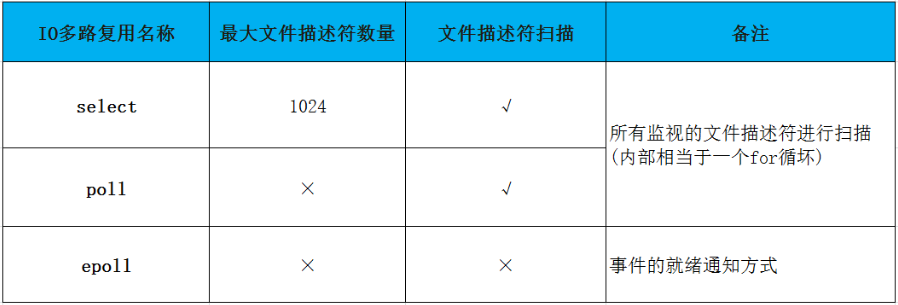
直到Linux2.6才出现了由内核直接支持的实现方法，那就是epoll，它几乎具备了之前所说的一切优点，被公认为Linux2.6下性能最好的多路I/O就绪通知方法。

　　 epoll可以同时支持水平触发和边缘触发（Edge Triggered，只告诉进程哪些 文件描述符刚刚变为就绪状态，它只说一遍，如果我们没有采取行动，那么它将不 会再次告知，这种方式称为边缘触发），理论上边缘触发的性能要更高一些，但是 代码实现相当复杂。

　　 epoll同样只告知那些就绪的文件描述符，而且当我们调用epoll\_wait()获得 就绪文件描述符时，返回的不是实际的描述符，而是一个代表就绪描述符数量的值， 你只需要去epoll指定的一个数组中依次取得相应数量的文件描述符即可，这里也 使用了内存映射（mmap）技术，这样便彻底省掉了这些文件描述符在系统调用 时复制的开销。

　　 另一个本质的改进在于epoll采用基于事件的就绪通知方式。在select/poll 中，进程只有在调用一定的方法后，内核才对所有监视的文件描述符进行扫描，而 epoll事先通过epoll\_ctl()来注册一个文件描述符，一旦基于某个文件描述符就绪 时，内核会采用类似callback的回调机制，迅速激活这个文件描述符，当进程调 用epoll\_wait()时便得到通知。

4、sellect、poll、epoll三者的区别



二、select IO多路复用

Python的select()方法直接调用操作系统的IO接口，它监控sockets,open files, and pipes(所有带fileno()方法的文件句柄)何时变成readable 和writeable, 或者通信错误，select()使得同时监控多个连接变的简单，并且这比写一个长循环来等待和监控多客户端连接要高效，因为select直接通过操作系统提供的C的网络接口进行操作，而不是通过Python的解释器。

select目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点。select的一 个缺点在于单个进程能够监视的文件描述符的数量存在最大限制，在Linux上一般为1024，可以通过修改宏定义甚至重新编译内核的方式提升这一限制，但是这样会造成效率的降低

1、select语法：select(rlist, wlist, xlist, timeout=None)

select()方法接收并监控3个通信列表， 第一个rlist监控所有要进来的输入数据，第二个wlist是监控所有要发出去的输出数据,第三个监控异常错误数据，第四个设置指定等待时间，如果想立即返回，设为null即可，最后需要创建2个列表来包含输入和输出信息来传给select()，让select方法通过内核去监控,然后生成三个实例。

#建立两个列表，比如想让内核去检测50个连接，需要传给它一个列表，就是这个inputs（列表中里面存放的是需要被内核监控的链接），然后交给select，就相当于交给内核了  
inputs = [server,] #输入列表，监控所有输入数据  
  
outputs = [] #输入列表，监控所有输出数据  
  
#把两个列表传给select方法通过内核去监控，生成三个实例  
readable,writeable,exceptional = select.select(inputs,outputs,inputs) # 这里select方法的第三个参数同样传入input列表是因为，input列表中存放着所有的链接，比如之前放入的50被监控链接中有5个断了，出现了异常，就会输入到exceptional里面，但这5链接本身是放在inputs列表中

Select服务端代码示例：

import select, socket, queue  
  
server = socket.socket()  
server.bind((**"localhost"**, 9000))  
server.listen(1000)  
server.setblocking(False) # 设置为非阻塞  
msg\_dic = dict() # 定义一个队列字典  
inputs = [server, ] # 由于设置成非阻塞模式，accept和recive都不阻塞了，没有值就会报错，因此最开始需要最开始需要监控服务端本身，等待客户端连接  
outputs = []  
while True:  
 # exceptional表示如果inputs列表中出现异常，会输出到这个exceptional中  
 readable, writeable, exceptional = select.select(inputs, outputs, inputs) # 如果没有任何客户端连接，就会阻塞在这里   
 for r in readable: # 没有个r代表一个socket链接  
 if r is server: # 如果这个socket是server的话，就说明是是新客户端连接了  
 conn, addr = r.accept() # 新连接进来了,接受这个连接，生成这个客户端实例  
 print(**"来了一个新连接"**, addr)  
 inputs.append(conn) # 为了不阻塞整个程序,我们不会立刻在这里开始接收客户端发来的数据, 把它放到inputs里, 下一次loop时,这个新连接  
 # 就会被交给select去监听  
 msg\_dic[conn] = queue.Queue() # 初始化一个队列，后面存要返回给这个客户端的数据  
 else: # 如果不是server，就说明是之前建立的客户端来数据了  
 data = r.recv(1024)  
 print(**"收到数据："**, data)  
 msg\_dic[r].put(data) # 收到的数据先放到queue里,一会返回给客户端  
 outputs.append(r) # 为了不影响处理与其它客户端的连接 , 这里不立刻返回数据给客户端  
 # r.send(data)  
 # print("send done....")  
 for w in writeable: # 要返回给客户端的链接列表  
 data\_to\_client = msg\_dic[w].get()  
 w.send(data\_to\_client) # 返回给客户端的源数据  
 outputs.remove(w) # 确保下次循环的时候writeable,不返回这个已经处理完的这个连接了  
  
 for e in exceptional: # 处理异常的连接  
 if e in outputs: # 因为e不一定在outputs，所以先要判断  
 outputs.remove(e)  
 inputs.remove(e) # 删除inputs中异常连接  
 del msg\_dic[e] # 删除此连接对应的队列

三、epoll IO多路复用

epoll的方式，这种效率更高，但是这种方式在Windows下不支持，在Linux是支持的，selectors模块就是默认使用就是epoll，但是如果在windows系统上使用selectors模块，就会找不到epoll，从而使用select。

1. selectors语法：

#定义一个对象  
sel = selectors.DefaultSelector()  
  
#注册一个事件  
sel.register(server,selectors.EVENT\_READ,accept) #注册事件，只要来一个连接就调accept这个函数，就相当于之前select的用法，sel.register(server,selectors.EVENT\_READ,accept) == inputs=[server,]，readable,writeable,exceptional = select.select(inputs,outputs,inputs)意思是一样的。

Selectors代码示例：

import selectors, socket  
  
sel = selectors.DefaultSelector()  
  
  
def accept(sock, mask):  
 "接收客户端信息实例"  
 conn, addr = sock.accept()  
 print(**"accepted"**, conn, **'from'**, addr)  
 conn.setblocking(False)  
 sel.register(conn, selectors.EVENT\_READ, read) # 新连接注册read回调函数  
  
  
def read(conn, mask):  
 "接收客户端的数据"  
 data = conn.recv(1024)  
 if data:  
 print(**"echoing"**, repr(data), **'to'**, conn)  
 conn.send(data)  
 else:  
 print(**"closing"**, conn)  
 sel.unregister(conn)  
 conn.close()  
  
  
server = socket.socket()  
server.bind((**'localhost'**, 9999))  
server.listen(500)  
server.setblocking(False)  
sel.register(server, selectors.EVENT\_READ, accept) # 注册事件，只要来一个连接就调accept这个函数,  
# sel.register(server,selectors.EVENT\_READ,accept) == inputs=[server,]  
  
while True:  
 events = sel.select() # 这个select,看起来是select，有可能调用的是epoll，看你操作系统是Windows的还是Linux的  
 # 默认阻塞，有活动连接就返回活动连接列表  
 print(**"事件："**, events)  
 for key, mask in events:  
 callback = key.data # 相当于调accept了  
 callback(key.fileobj, mask) # key.fileobj=文件句柄

客户端代码：

import socket, sys  
  
messages = [b'This is the message. ',  
 b'It will be sent ',  
 b'in parts.',  
 ]  
server\_address = (**'localhost'**, 9999)  
  
# 创建100个 TCP/IP socket实例  
socks = [socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) for i in range(100)]  
  
# 连接服务端  
print(**'connecting to %s port %s'** % server\_address)  
for s in socks:  
 s.connect(server\_address)  
  
for message in messages:  
  
 # 发送消息至服务端  
 for s in socks:  
 print(**'%s: sending "%s"'** % (s.getsockname(), message))  
 s.send(message)  
  
 # 从服务端接收消息  
 for s in socks:  
 data = s.recv(1024)  
 print(**'%s: received "%s"'** % (s.getsockname(), data))  
 if not data:  
 print(sys.stderr, **'closing socket'**, s.getsockname())

1. **什么是防火墙？**

[防火墙](https://baike.baidu.com/item/%E9%98%B2%E7%81%AB%E5%A2%99/52767" \t "https://baike.baidu.com/item/%E9%98%B2%E7%81%AB%E5%A2%99/_blank)（Firewall），也称防护墙，是由Check Point创立者Gil Shwed于1993年发明并引入国际互联网（US5606668（A）1993-12-15）。防火墙是位于内部网和外部网之间的屏障，它按照系统管理员预先定义好的规则来控制数据包的进出。防火墙是系统的第一道防线，其作用是防止非法用户的进入。

1. **什么是select？select，poll，epoll模型的区别是什么？**

答案见第十题。

1. **详细解释水平触发和边缘触发**

在linux的IO多路复用中有水平触发,边缘触发两种模式,这两种模式的区别如下:

水平触发:如果文件描述符已经就绪可以非阻塞的执行IO操作了,此时会触发通知.允许在任意时刻重复检测IO的状态,没有必要每次描述符就绪后尽可能多的执行IO.select,poll就属于水平触发.

    边缘触发:如果文件描述符自上次状态改变后有新的IO活动到来,此时会触发通知. 在收到一个IO事件通知后要尽可能多的执行IO操作,因为如果在一次通知中没有执行 完IO那么就需要等到下一次新的IO活动到来才能获取到就绪的描述符.信号驱动式IO 就属于边缘触发.

    epoll既可以采用水平触发,也可以采用边缘触发.

    大家可能还不能完全了解这两种模式的区别,我们可以举例说明:一个管道收到了 1kb的数据,epoll会立即返回,此时读了512字节数据,然后再次调用epoll.这时如果是 水平触发的,epoll会立即返回,因为有数据准备好了.如果是边缘触发的不会立即返回,因 为此时虽然有数据可读但是已经触发了一次通知,在这次通知到现在还没有新的数据到 来,直到有新的数据到来epoll才会返回,此时老的数据和新的数据都可以读取到(当然是 需要这次你尽可能的多读取).

下面我们还从电子的角度来解释一下:

    水平触发:也就是只有高电平(1)或低电平(0)时才触发通知,只要在这两种状态就能 得到通知.上面提到的只要有数据可读(描述符就绪)那么水平触发的epoll就立即返回.

    边缘触发:只有电平发生变化(高电平到低电平,或者低电平到高电平)的时候才触发 通知.上面提到即使有数据可读,但是没有新的IO活动到来,epoll也不会立即返回.

1. **详细解释网络的同步与异步。**

使用同步链接，当同步连接请求数据时，当数据还未请求成功之前，用户界面做任何操作都是无效的，都是不会响应的。只有当请求数据完毕之后，才会响应用户交互，所以通常会卡死主线程

异步连接：当异步请求时，会有单独的子线程去请求数据，而主线程依然响应处理用户交互，所以此时用户交互得到处理，用户流畅操作，用户体验比较好，所以开发过程中最多的还是异步链接。

1. **简单分析HTTP协议的请求报文和响应报文**

1.HTTP协议与报文简介

 HTTP（hypertext transport protocol），即超文本传输协议。这个协议详细规定了浏览器和万维网服务器之间互相通信的规则。

而客户端与服务端通信时传输的内容我们称之为报文。

  HTTP就是一个通信规则，这个规则规定了客户端发送给服务器的报文格式，也规 定了服务器发送给客户端的报文格式。

客户端发送给服务器的称为”请求报文“，服务器发送给客户端的称为”响应报文“。

 2.报文格式

   报文格式： 报文首部  空行  报文主体

    2.1 请求报文 ：请求首部   
  get请求

  GET /Hello/index.jsp HTTP/1.1              请求首行：请求方式 请求资源【url】 请求协议/版本号  
  Accept: \*/\*                                              请求头部：\*/\*表示客户端可 以接受的文档类型，\*/\*表示的是可以接受任意类型  
  Accept-Language: zh-CN                       Accept-Language:表示的是当 前浏览器所在的地区，zh-CN表示是客户端用的语言简体中文，en-US表示的是 美式英语！  
  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center  PC 6.0; .NET4.0C; .NET4.0E)  
  Accept-Encoding: gzip,deflate     浏览器向服务器发送请求的时候，会告诉服 务器端浏览器可以接受的压缩格式！  
  Host: localhost:8080                   是目标主机的ip地址和端口号！  
 Connection: Keep-Alive              表示的是浏览器和服务器的通信连接： Keep-Alive 表示会话保持！默认会话连接时间是3000ms!  
  Cookie: JSESSIONID=C55836CDA892D9124C03CF8FE8311B15   是位于浏 览器端的，和服务器通信使用！  
  注意：get请求没有空行和请求体！  
 post请求(大体与get相同）  
 POST /Hello/target.html HTTP/1.1  
 Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap, \*/\*  
 Referer: http://localhost:8080/Hello/  
 Accept-Language: zh-CN  
 User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)  
 Content-Type: application/x-www-form-urlencoded  
 Accept-Encoding: gzip, deflate  
 Host: localhost:8080  
 Content-Length: 14  
 Connection: Keep-Alive  
 Cache-Control: no-cache  
 Cookie: JSESSIONID=774DA38C1B78AE288610D77621590345

2.2 响应报文： 响应首部 空行 响应主体

响应报文是由响应首部[响应首行和响应头部]、空行、响应体

HTTP/1.1 200 OK                                                      响应首行：通 信协议/版本号 响应状态码   
 Server: Apache-Coyote/1.1                                      服务器：apache  
 Content-Type: text/html;charset=UTF-8                  响应体编码类型： Content-Type  
 Content-Length: 274                                              响应体长度： Content-Length  
 Date: Tue, 07 Apr 2015 10:08:26 GMT                 响应时间:Date

响应状态码：  
 200：表示的是响应成功  
 302：表示的是重定向  
 404：请求的url错误！  
 500：服务器端的错误！

1. **简单解释HTTP的长连接和短连接**

HTTP1.1规定了默认保持长连接（HTTP persistent connection ，也有翻译为持久连接），数据传输完成了保持TCP连接不断开（不发RST包、不四次握手），等待在同域名下继续用这个通道传输数据；相反的就是短连接。

　 HTTP首部的Connection: Keep-alive是HTTP1.0浏览器和服务器的实验性扩展，当前的HTTP1.1 RFC2616文档没有对它做说明，因为它所需要的功能已经默认开启，无须带着它，但是实践中可以发现，浏览器的报文请求都会带上它。如果HTTP1.1版本的HTTP请求报文不希望使用长连接，则要在HTTP请求报文首部加上Connection: close。《HTTP权威指南》提到，有部分古老的HTTP1.0 代理不理解Keep-alive，而导致长连接失效：客户端-->代理-->服务端，客户端带有Keep-alive，而代理不认识，于是将报文原封不动转给了服务端，服务端响应了Keep-alive，也被代理转发给了客户端，于是保持了“客户端-->代理”连接和“代理-->服务端”连接不关闭，但是，当客户端第发送第二次请求时，代理会认为当前连接不会有请求了，于是忽略了它，长连接失效。书上也介绍了解决方案：当发现HTTP版本为1.0时，就忽略Keep-alive，客户端就知道当前不该使用长连接。其实，在实际使用中不需要考虑这么多，很多时候代理是我们自己控制的，如Nginx代理，代理服务器有长连接处理逻辑，服务端无需做patch处理，常见的是客户端跟Nginx代理服务器使用HTTP1.1协议&长连接，而Nginx代理服务器跟后端服务器使用HTTP1.0协议&短连接。

    在实际使用中，HTTP头部有了Keep-Alive这个值并不代表一定会使用长连接，客户端和服务器端都可以无视这个值，也就是不按标准来，譬如我自己写的HTTP客户端多线程去下载文件，就可以不遵循这个标准，并发的或者连续的多次GET请求，都分开在多个TCP通道中，每一条TCP通道，只有一次GET，GET完之后，立即有TCP关闭的四次握手，这样写代码更简单，这时候虽然HTTP头有Connection: Keep-alive，但不能说是长连接。正常情况下客户端浏览器、web服务端都有实现这个标准，因为它们的文件又小又多，保持长连接减少重新开TCP连接的开销很有价值。

     以前使用[libcurl做的上传](http://www.cnblogs.com/cswuyg/p/3185164.html" \t "https://www.cnblogs.com/cswuyg/p/_blank)/下载，就是短连接，抓包可以看到：1、每一条TCP通道只有一个POST；2、在数据传输完毕可以看到四次握手包。只要不调用curl\_easy\_cleanup，curl的handle就可能一直有效，可复用。这里说可能，因为连接是双方的，如果服务器那边关掉了，那么我客户端这边保留着也不能实现长连接。

    如果是使用windows的WinHTTP库，则在POST/GET数据的时候，虽然我关闭了句柄，但这时候TCP连接并不会立即关闭，而是等一小会儿，这时候是WinHTTP库底层支持了跟Keep-alive所需要的功能：即便没有Keep-alive，WinHTTP库也可能会加上这种TCP通道复用的功能，而其它的网络库像libcurl则不会这么做。以前观察过[WinHTTP库不会及时断开TCP连接](http://www.cnblogs.com/cswuyg/p/3516417.html" \t "https://www.cnblogs.com/cswuyg/p/_blank)。

1. **TCP会自动断开连接嘛？TCP服务器如何判断客户端已断开连接？**

TCP的保活定时器能够保证TCP连接一直保持，但是TCP的保活定时器不是每个TCP/IP协议栈就实现了，因为RFC并不要求TCP保活定时器一定要实现。再补充一点，TCP保活定时器的功能是每隔一个时间t (可能t=2小时，依赖具体的实现。）发送保活分组。

1. **什么是TCP流量控制？**

流量控制——滑动窗口

TCP采用大小可变的滑动窗口进行流量控制，窗口大小的单位是字节。

发送窗口在连接建立时由双方商定。但在通信的过程中，接收端可根据自己的资源情况，随时动态地调整对方的发送窗口上限值(可增大或减小)。

为什么要设置窗口？

我们可以把窗口理解为缓冲区（但是有些窗口和缓冲区又不太一样）。

如果没有这些“窗口”，那么TCP没发送一段数据后都必须等到接收端确认后才能发送下一段数据，这样做的话TCP传输的效率实在是太低了。

解决的办法就是在发送端等待确认的时候继续发送数据，假设发送到第X个数据段是收到接收端的确认信息，如果X在可接受的范围内那么这样做也是可接受的。这就是窗口（缓冲区）引入的缘由。

1. **GET和POST请求的区别？**

GET在浏览器回退时是无害的，而POST会再次提交请求。

GET产生的URL地址可以被Bookmark，而POST不可以。

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST么有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中。

1. **http与https的区别？**

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

1. **cookie和session的区别是什么？**

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

1. **详细说明：当你在浏览器输入一个地址后，回车，发生了什么？**

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

1. **http中的缓存是如何实现的？**

答案参见链接：<https://www.cnblogs.com/exhuasted/p/7814729.html>

1. **什么是域名解析？**

[域名](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D" \t "https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E8%A7%A3%E6%9E%90/_blank)解析是把域名指向网站空间IP，让人们通过注册的域名可以方便地访问到网站的一种服务。[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80" \t "https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E8%A7%A3%E6%9E%90/_blank)是网络上标识站点的数字地址，为了方便记忆，采用域名来代替IP地址标识站点地址。域名解析就是域名到IP地址的转换过程。域名的解析工作由[DNS服务器](https://baike.baidu.com/item/DNS%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E8%A7%A3%E6%9E%90/_blank)完成。

域名解析也叫[域名指向](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E6%8C%87%E5%90%91/2779846" \t "https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D%E8%A7%A3%E6%9E%90/_blank)、服务器设置、域名配置以及反向IP登记等等。说得简单点就是将好记的域名解析成IP，服务由DNS服务器完成，是把域名解析到一个IP地址，然后在此IP地址的主机上将一个子目录与域名绑定。

互联网中的地址是数字的IP地址，域名解析的作用主要就是为了便于记忆。

1. **如何修改本地hosts文件？**

统一思路都为先修改文件权限(嫌麻烦直接修改为最高权限),再使用记事本打开修改内容保存即可。具体的windows下与linux下操作方式略有不同。