1.先　　天　　　　：

场：

- 　试官：你说　下TCP的三次握手
- 我：第　次Client将SYN置1.....　第二次Server收.....　第三次.....
- 　试官：很　背吧?
- 我：.....是啊，很　，要不我在和你说说四次挥手?
- 　试官：别了别了回去等　知吧.....
- 我："....."

二场： `心里憋了一万个草泥马来到的第二家`

- ....
- 　试官：你说　下TCP的三次握手
- 我（`心里在想，还来?`）：没什么好说的，就是为了保持　次网络　信交互正常
- 　试官：你能说的清楚　点吗?
- 我：就等于是你在不认识我的情况下打我的电话让我来　试
- 　试官：“　了”，好像是　么回事
- 　试官：你说　下TCP的四次挥手
- 我：等于是我在上家公司　职了
- 　试官：“想了　下”，能不能说的清楚　点吗?
- 我：我找老板办理离职，老板说可以，老板接着给我办理离职，我才可以
- 　试官：有　理!
- 　试官：你说　下TCP和UDP的区别吧
- 我：TCP等于和　生人打电话处理事情，UDP等于发广播
- 　试官：“...”有　理
- 　试官：你期望　　多少
- 我：15K
- 　试官：下周　有时　入职?
- 我：....

接着进入正题

2.什么是

- 网络编程的本　是多台计算机之　的数据交换　数据传　本　没有多大的　度，不就是把　个设备中的数据发　给其他设备，然后接受另外　个设备反　的数据　现在的网络编程基本上　是基于求/响应方式的，也就是　个设备发　　求数据给另外　个，然后接收另　个设备的反　　在网络编程中，发　　接程序，也就是发　第　次　求的程序，被称作客户端(Client)，等待其他程序接的程序被称作服务器(Server)　客户端程序可以在　要的时候启动，而服务器为了能够时刻相应　接，则　要　直启动
- 例如以打电话为例，　先拨号的人类似于客户端，接听电话的人必　保持电话畅　类似于服务器　接　旦建立以后，就客户端和服务器端就可以　行数据传　了，而且两者的　份是等价的　在些程序中，程序既有客户端功能也有服务器端功能，最常见的　件就是QQ　微信　类　件了

3.　　中两个主

1. 　个是如何准确的定位网络上　台或多台主机，
2. 另　个就是找到主机后如何可　　效的　行数据传

- 在TCP/IP协议中IP层主要　　网络主机的定位，数据传　的　由，由IP地址可以唯　地确定Internet上的　台主机
- 而TCP层则提供　向应用的可　（TCP）的或　可　（UDP）的数据传　机制，　是网络编程的主要对　，　般不　要关心IP层是如何处理数据的
- 目前　为流行的网络编程模型是客户机/服务器（C/S）结构　即　信双方　方作为服务器等待客户提出　求并予以响应　客户则在　要服务时向服务器提 出申　　服务器　般作为守护　程始终行，监听网络端口，　旦有客户　求，就会启动　个服务　程来响应该客户，同时自己继续监听服务端口，使后来的客户也 能及时得到服务

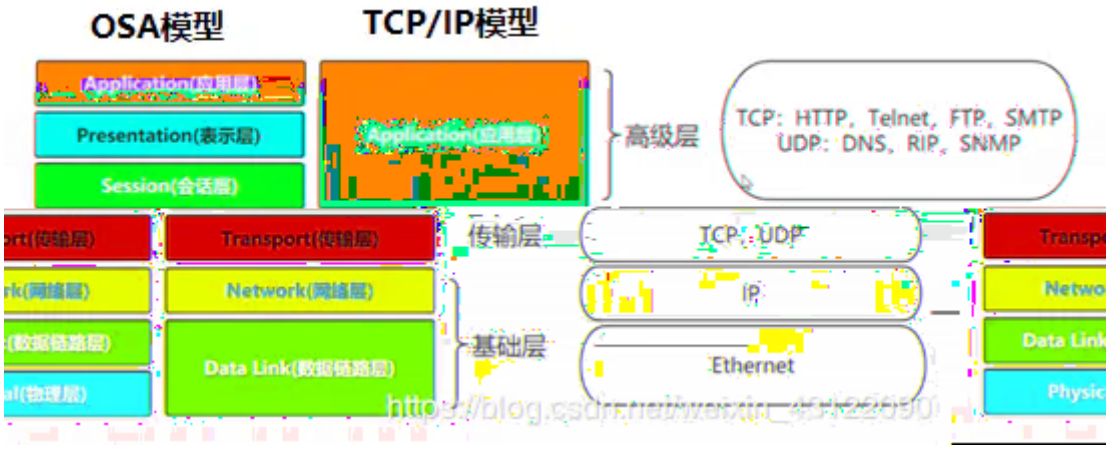
4.　　协　是什么

- 在计算机网络要做到井井有条的交换数据，就必　　守　些事先约定好的规则，比如交换数据的格式　是否　要发　　个应答信息　　些规则被称为网络协议

5.为什么　对　　协　分层

- 简化　　度和复杂度　由于各层之　独立，我们可以分割大　　为小
- 灵活性好　当其中　层的技术变化时，只要层　接口关系保持不变，其他层不受影响
- 易于实现和维护
- 促　标准化工作　分开后，每层功能可以相对简单地被描

6.　　机　　体　　构



- **OSI参　模型**
  - OSI（Open System Interconnect），即开放式系统互联　　般　叫OSI参考模型，是ISO（国　标准化组织）组织在1985年研究的网络互　模型　ISO为了更好的使网络应用更为普及，推出了OSI参考模型，　样所有的公司　按照统　的标准来指定自己的网络，就可以互　互联了
  - OSI定义了网络互　的七层框架（物理层　数据　　层　网络层　传　层　会话层　表示层　应用层）

OSI七层模型	功能	对应的网络协议	TCP/IP四层概念模型
应用层	文件传输，文件管理，电子邮件的信息处理——apdu	HTTP、TFTP，FTP，NFS，WAIS、SMTP	
表示层	确保一个系统的应用层发送的消息可以被另一个系统的应用层读取，编码转换，数据解析，管理数据的解密和加	Telnet，Rlogin，SNMP，Gopher	
——ppdu			密，最小单位
-层协议中，可以解决节点 管理问题。包括通信连接 会话过程通信连接的畅 间的对话，决定通信是否 通信终端是决定从何处重新 位——spdu	SMTP，DNS	应用层 会话层	负责在网络 止通信，在一 连接的协调和 的建立，保持 通，两节点之 被终端一斤退 发送，最小单
输数据的协议和端口。传输 行流量控制，或是根据接收 的快慢程度，规定适当的发 传输效率及能力的问题	TCP，UDP	传输层 传输层	定义一些传输 协议同时进行 方接收数据的 送顺序，解
——tpdu			
组传 ）	IP，ICMP，ARP，RARP，AKP，UUCP	网络层 网络层	控制子网的运行，如逻辑编址，分组 输，路由选择最小单位——分组（包 报文
检 接 址	FDDI，Ethernet，		主要是对物理层传输的比特流包装， 测保证数据传输的可靠性，将物理层 收的数据进行MAC（媒体访问控制）
物理寻址，交换机就处在这一层，最小 标准传输单位			
物理层	定义物理设备的标准，主要对物理连接方式，电气特性，机械特性等制定统一标准，传输比特流，因此最小的传输单位——位（比特流）	IEEE 802.1A，IEEE 802.2到IEEE 802.	

- **TCP/IP参　模型**
- TCP/IP四层协议（数据　　层　网络层　传　层　应用层）

1. 应用层 应用层最　　用户的　层，是为计算机用户提供应用接口，也为用户直接提供各种网络服务　我们常见应用层的网络服务协议有：HTTP，HTTPS，FTP，TELNET等
2. 传　层 建立了主机端到端的　接，传　层的作用是为上层协议提供端到端的可　和　明的数据传　服务，包括处理差　控制和流　控制等　　该层向　层屏蔽了下层数据　信的细节，使　层用户看到的只是在两个传　实体　的　条主机到主机的　可由用户控制和设定的　可　的数据　　我们　常说的，TCP UDP就是在　　层　端口号既是　　的"端"
3. 网络层 本层　　IP寻址来建立两个节点之　的　接，为源端的　　层　来的分组，　择合的　由和交换节点，正确无误地按照地址传　给目的端的　　层　就是　常说的IP层　　层就是我们经常说的IP协议层　IP协议是Internet的基础
4. 数据　　层　　些规程或协议来控制　些数据的传　，以保证被传　数据的正确性　实现　　些规程或协议的硬件和　件加到物理线　，　样就构成了数据　　，

1 TCP / UDP

``

1.1 什么是TCP/IP和UDP

- TCP/IP即传　控制/网络协议，是　向　接的协议，发　数据前要先建立　接(发　方和接收方的成对的两个之　必　建 立　接)，TCP提供可　的服务，也就是说，　　TCP　接传　的数据不会丢失，没有　复，并且按　序到
- UDP它是属于TCP/IP协议族中的　种　是无　接的协议，发　数据前不　要建立　接，是没有可　性的协议　因为不　要建立　接所以可以在在网络上以任何可能的　径传　，因此能否到　目的地，到　目的地的时　以及内容的正确性　是不能被保证的

1.2 TCP与UDP区别：

- TCP是　向　接的协议，发　数据前要先建立　接，TCP提供可　的服务，也就是说，　　TCP接传　的数据不会丢失，没有　复，并且按　序到　；
- UDP是无　接的协议，发　数据前不　要建立　接，是没有可　性；
- TCP　信类似于于要打个电话，接　了，确认　份后，才开始　行　行；
- UDP　信类似于学校广播，　着广播播报直接　行　信
- TCP只支持点对点　信，UDP支持　对　　对多　多对　　多对多；
- TCP是　向字节流的，UDP是　向报文的；　向字节流是指发　数据时以字节为单位，　个数据包可以拆分成若干组　行发　，而UDP　个报文只能　次发完
- TCP　　开　（20字节）比UDP　　开　（8字节）要大
- UDP 的主机不　要维持复杂的　接状态表

1.3 TCP和UDP　应　场景：

- 对某些实时性要求比　　的情况使用UDP，比如游戏，媒体　信，实时直播，即使出现传　　误也可以容忍；其它大　分情况下，HTTP　是用TCP，因为要求传　的内容可　，不出现丢失的情况

1.4 形容　下TCP和UDP

- **TCP　信可　作打　　：**

  三(拨了个号码)：　，是王五吗？王五：哎，您　啊？　三：我是　三，我想给你说点事儿，你现在方便吗？王五：哦，我现在方便，你说吧　　：　我说了啊？　：你说吧　（　接建立了，接下来就是说正事了...)

- **UDP　信可　为学校　　广播：**

  播　室：　　！全体操场　合

1.5　　在TCP 或UDP　应　层协　分析

- 行在TCP协议上的协议：
  - HTTP（Hypertext Transfer Protocol，　文本传　协议），主要用于普　浏览
  - HTTPS（HTTP over SSL，安全　文本传　协议）,HTTP协议的安全版本
  - FTP（File Transfer Protocol，文件传　协议），用于文件传
  - POP3（Post Office Protocol, version 3，　局协议），收　件用
  - SMTP（Simple Mail Transfer Protocol，简单　件传　协议），用来发　电子　件
  - TELNET（Teletype over the Network，网络电传），　　个终端（terminal）登　到网络
  - SSH（Secure Shell，用于替代安全性差的TELNET），用于加密安全登　用
- 行在UDP协议上的协议：
  - BOOTP（Boot Protocol，启动协议），应用于无盘设备
  - NTP（Network Time Protocol，网络时　协议），用于网络同步
  - DHCP（Dynamic Host Configuration Protocol，动态主机　置协议），动态　置IP地址
- 行在TCP和UDP协议上：

- DNS（Domain Name Service，域名服务），用于完成地址查找， 件 发等工作
- ECHO（Echo Protocol，回绕协议），用于查 及测 应答时 （ 行在TCP和UDP协议上）
- SNMP（Simple Network Management Protocol，简单网络管理协议），用于网络信息的收 和网络管理
- DHCP（Dynamic Host Configuration Protocol，动态主机 置协议），动态 置IP地址
- ARP（Address Resolution Protocol，地址解析协议），用于动态解析以太网硬件的地址

什么是ARP协 (Address Resolution Protocol)?

- **ARP协 完成了IP地址与物理地址 映射** 每 个主机 设有 个 ARP 缓存， 有**所在局域** 上的各主机和 由器的 IP 地址到硬件地址的映射表 当源主机要发 数据包到目的主机时，会先检查自己的ARP 缓存中有没有目的主机的MAC地址，如果有，就直接将数据包发到个MAC地址，如果没有，就向**所在 局域** 发 个ARP 求的广播包（在发 自己的 ARP 求时，同时会带上自己的 IP 地址到硬件地址的映射），收到 求的主机检查自己的IP地址和目的主机的IP地址是否 致，如果 致，则先保存源主机的映射到自己的ARP缓存，然后给源主机发个ARP响应数据包 源主机收到响应数据包之后，先添加目的主机的IP地址与MAC地址的映射，再 行数据传 如果源主机 直没有收到响应，表示ARP查询失
- 如果所要找的主机和源主机不在同 个局域网上， 么就要 ARP 找到 个位于本局域网上的某个 由器的硬件地址，然后把分组发 给 个 由器，让 个 由器把分组 发给下 个网络剩下的工作就由下 个网络来做

什么是NAT (Network Address Translation, 地址 换)?

- 用于解决内网中的主机要和因特网上的主机 信 由NAT 由器将主机的本地IP地址 换为全球IP地址，分为 态 换（ 换得到的全球IP地址固定不变）和动态NAT 换

从 入址到 得 ?

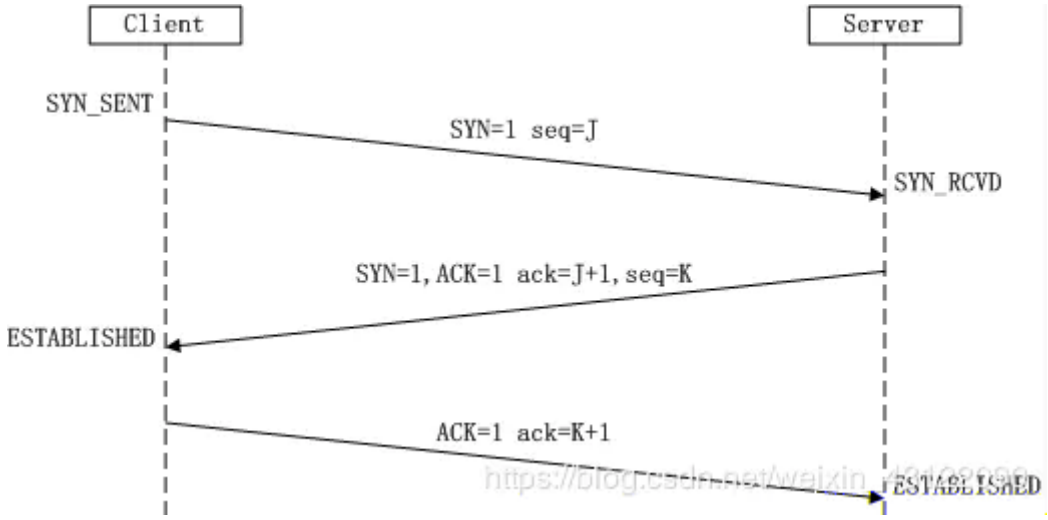
1. 浏览器查询 DNS，获取域名对应的IP地址:具体 程包括浏览器搜索自 的DNS缓存 搜索操作系统的DNS缓存 取本地的Host文件和向本地DNS服务器 行查询等 对于向本地DNS服务器 行查询，如果要查询的域名包含在本地 置区域 源中，则 回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则 用 个IP地址映射，完成域名解析（此解析不具有权威性） 如果本地域名服务器并未缓存该网址映射关系， 么将根据其设置发 归查询或者 代查询；
2. 浏览器获得域名对应的IP地址以后，浏览器向服务器 求建立 接，发 三次握手；
3. TCP/IP 接建立 来后，浏览器向服务器发 HTTP 求；
4. 服务器接收到 个 求，并根据 径参数映射到特定的 求处理器 行处理，并将处理结果及相应的视图 回给浏览器；
5. 浏览器解析并 染视图，若 到对js文件 css文件及图片等 态 源的引用，则 复上 步 并向服务器 求 些 源；
6. 浏览器根据其 求到的 源 数据 染 ，最终向用户呈现 个完整的

1.6 TCP 三次握手

1.6.1 什么是TCP 三次握手

- 在网络数据传 中，传 层协议TCP是要建立 接的可 传 ，TCP建立 接的 程，我们称为三次握手

1.6.2 三次握手 具体



1. 第　次握手：Client将SYN置1，　机产生　个初始序列号seq发　给Server，　入SYN_SENT状态；
2. 第二次握手：Server收到Client的SYN=1之后，知　客户端　求建立　接，将自己的SYN置1，ACK置1，产生　个acknowledge number=sequence number+1，并　机产生　个自己的初始序列号，发　给客户端；　入SYN_RCVD状态；
3. 第三次握手：客户端检查acknowledge number是否为序列号+1，ACK是否为1，检查正确之后将自己的ACK置为1，产生　个acknowledge number=服务器发的序列号+1，发　给服务器；　入ESTABLISHED状态；服务器检查ACK为1和acknowledge number为序列号+1之后，也　入ESTABLISHED状态；完成三次握手，　接建立

- 简单来说就是：
    1. 客户端向服务端发　SYN
    2. 服务端　回SYN,ACK
    3. 客户端发　ACK

1.6.3　现实理　三次握手　具体

- 三次握手的目的是建立可　的　信信　，主要的目的就是双方确认自己与对方的发　与接收机能正常

1. 第　次握手：客户什么　不能确认；服务器确认了对方发　正常
2. 第二次握手：客户确认了：自己发　　接收正常，对方发　　接收正常；服务器确认 了：自己接收正常，对方发　正常
3. 第三次握手：客户确认了：自己发　　接收正常，对方发　　接收正常；服务器确认 了：自己发　　接收正常，对方发　接收正常 所以三次握手就能确认双发收发功能　正常，缺　不可

1.6.4 建　　接可以两次握手吗？为什么?

- 不可以
- 因为可能会出现已失效的　接　求报文段又传到了服务器端　> client 发出的第　个　接　求报文段并没有丢失，而是在某个网络结点　时　的滞留了，以致延误到　接　放以后的某个时　才到 server　本来　是　个早已失效的报文段　但 server 收到此失效的　接　求报文段后，就误认为是 client 再次发出的　个新的　接　求　于是就向 client 发出确认报文段，同意建立　接　假设不　用“三次握手”，　么只要 server 发出确认，新的　接就建立了　由于现在 client 并没有发出建立　接的　求，因此不会理　server 的确认，也不会向 server 发　数据　但 server 却以为新的　　接已经建立，并　直等待 client 发来数据　　样，server 的很多　源就白白浪　掉了　用“三次握手”的办法可以　止上　现　发生　例如刚才　种情况，client 不会向 server 的确认发出确认　server 由于收不到确认，就知　client 并没有要求建立　接
- 而且，两次握手无法保证Client正确接收第二次握手的报文（Server无法确认Client是否收到），也无法保证Client和Server之　成功互换初始序列号

1.6.5 可以　四次握手吗？为什么?

- 个肯定可以　三次握手　可以保证　接成功了，何况是四次，但是会　低传　的效率

1.6.6　三次握手中，如果客户　ACK未　服务器，会　样?

- Server端：由于Server没有收到ACK确认，因此会每　3秒　发之前的SYN+ACK（　认　发五次，之后自动关　接　入CLOSED状态），Client收到后会　新传ACK给Server
- Client端，会出现两种情况：
  1. 在Server　行　时　发的　程中，如果Client向服务器发　数据，数据头　的ACK是为1的，所以服务器收到数据之后会　取 ACK number，　入 establish 状态
  2. 在Server　入CLOSED状态之后，如果Client向服务器发　数据，服务器会以RST包应答

1.6.7 如果已　建　了　接，但客户　出现了故　么办?

- 服务器每收到　次客户端的　求后　会　新复位　个计时器，时　常是设置为2小时，若两小时没有收到客户端的任何数据，服务器就会发　个探测报文段，以后每　75秒　发　次　若发　10个探测报文仍然没反应，服务器就认为客户端出了故　，接着就关　接

1.6.8 初始序列号是什么?

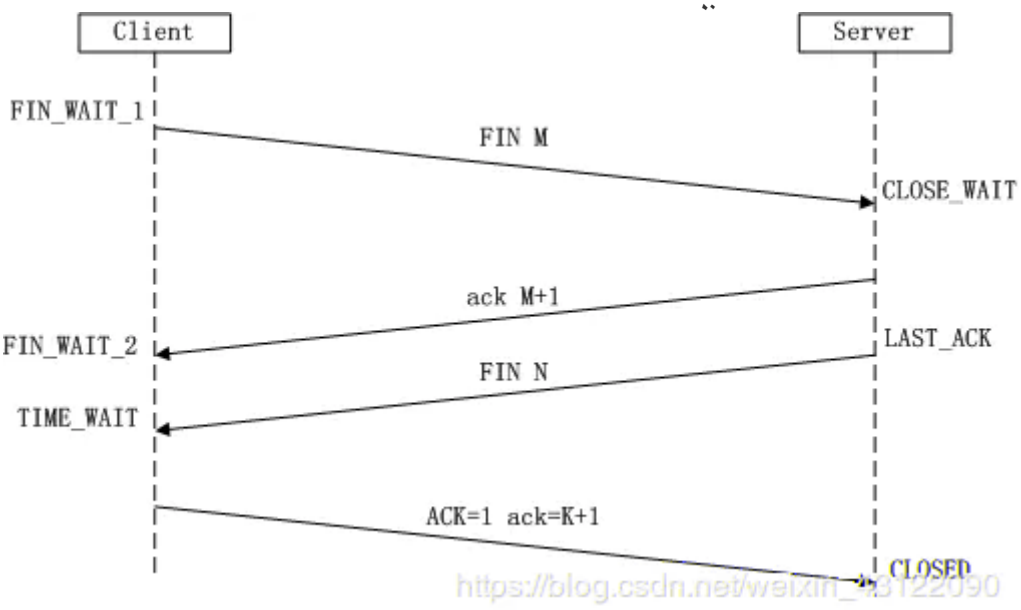
- TCP　接的　方A，　机　择　个32位的序列号（Sequence Number）作为发　数据的初始序列号（Initial Sequence Number，ISN），比如为1000，以该序列号为原点，对要传　的数据　行编号：1001　1002...三次握手时，把　个初始序列号传　给另　方B，以便在传　数据时，B可以确认什么样的数据编号是合法的；同时在　行数据传　时，A　可以确认B收到的每　个字节，如果A收到了B的确认编号（acknowledge number）是2001，就说明编号为1001-2000的数据已经被B成功接受

1.7 TCP　四次挥手

1.7.1 什么是TCP　四次挥手

- 在网络数据传　中，传　层协议断开　接的　程我们称为四次挥手

1.7.2 四次挥手　具体



1. 第　次挥手：Client将FIN置为1，发　个序列号seq给Server；　入FIN_WAIT_1状态；
2. 第二次挥手：Server收到FIN之后，发　个ACK=1，acknowledge number=收到的序列号+1；　入CLOSE_WAIT状态　此时客户端已经没有要发　的数据了，但仍可以接受服务器发来的数据
3. 第三次挥手：Server将FIN置1，发　个序列号给Client；　入LAST_ACK状态；

4. 第四次挥手：Client收到服务器的FIN后， 入TIME_WAIT状态；接着将ACK置1，发 个acknowledge number=序列号+1给服务器；服务器收到后，确认acknowledge number后，变为CLOSED状态，不再向客户端发 数据 客户端等待2*MSL（报文段最 寿命）时 后，也 入CLOSED状态 完成四次挥手

1.7.3 现实理 三次握手 具体 TCP 四次挥手

- 四次挥手断开 接是因为要确定数据全 传书完了

1. 客户与服务器交 结束之后，客户要结束此次会话，就会对服务器说：我要关 接了（第 次挥手）
2. 服务器收到客户的消息后说：好的，你要关 接了 （第二次挥手）
3. 然后服务器确定了没有话要和客户说了，服务器就会对客户说，我要关 接了 (第三次挥 手)
4. 客户收到服务器要结束 接的消息后说：已收到你要关 接的消息 (第四次挥手)，才关

1.7.4 为什么不 把服务器发 ACK和FIN合并 来，变成三次挥手（CLOSE_WAIT状 意义是什么）？

- 因为服务器收到客户端断开 接的 求时，可能 有 些数据没有发完， 时先回复ACK，表示接收到了断开 接的 求 等到数据发完之后再发FIN，断开服务器到客户端的数据传

1.7.5 如果 二次挥手时服务器`ACK没有 客户 ，会 样?

- 客户端没有收到ACK确认，会 新发 FIN 求

1.7.6 客户 TIME_WAIT状 意义是什么?

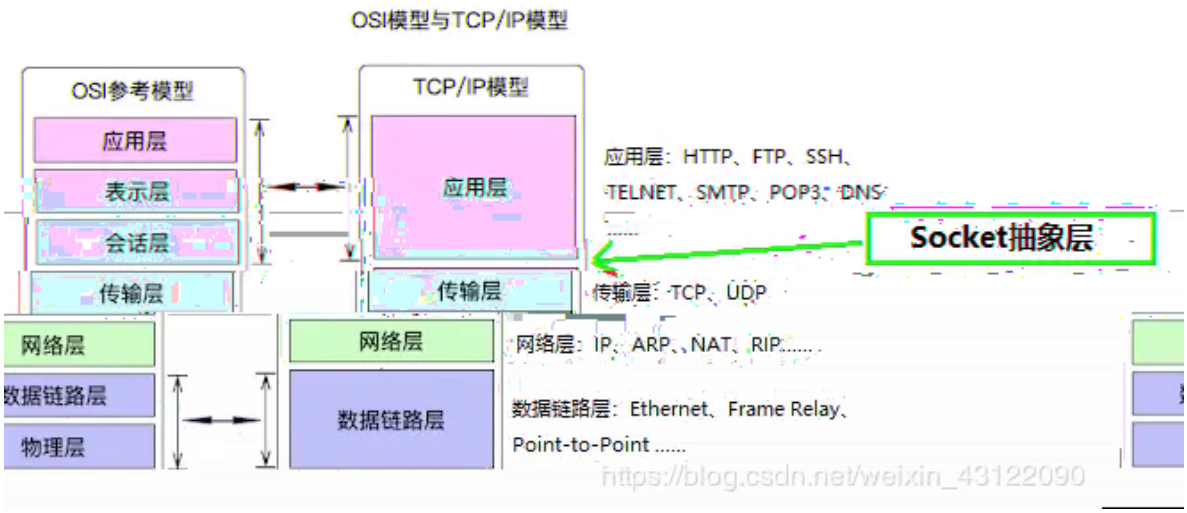
- 第四次挥手时，客户端发 给服务器的ACK有可能丢失，TIME_WAIT状态就是用来 发可能丢失的ACK报文 如果Server没有收到ACK，就会 发FIN，如果Client在2*MSL的时 内收到了FIN，就会 新发 ACK并再次等待2MSL， 止Server没有收到ACK而不断 发FIN MSL(Maximum Segment Lifetime)，指 个片段在网络中最大的存活时 ，2MSL就是 个发 和 个回复所 的最大时 如果直到2MSL，Client 没有再次收到FIN， 么Client推断ACK已经被成功接收，则结束TCP 接

2 Socket

1 什么是Socket

- 网络上的两个程序　　个双向的　讯　接实现数据的交换，　个双向　　的　端称为　个Socket　Socket　常用来实现客户方和服务方的　接　Socket是TCP/IP协议的　个十分流行的编程界　，　个Socket由　个IP地址和　个端口号唯　确定
- 但是，Socket所支持的协议种类也不光TCP/IP　UDP，因此两者之　是没有必然联系的　在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程
- socket　接就是所　的　　接，客户端和服务器　要互相　接，理论上客户端和服务器端　旦建立　　接将不会主动断掉的，但是有时候网络波动　是有可能的
- Socket偏向于底层　　般很少直接使用Socket来编程，框架底层使用Socket比　多，

2 socket属于　　个层



- Socket是应用层与TCP/IP协议族　信的中　　件抽　层，它是　组接口　在设计模式中，Socket其实就是　个外观模式，它把复杂的TCP/IP协议族　藏在Socket接口后　，对用户来说，　组简单的接口就是全　，让Socket去组织数据，以符合指定的协议

3 Socket

- 基于TCP：服务器端先初始化Socket，然后与端口绑定(bind)，对端口　行监听(listen)，　用accept　塞，等待客户端　接　在　时如果有个客户端初始化　个Socket，然后　接服务器(connect)，如果　接成功，　时客户端与服务器端的　接就建立了　客户端发　数据　求，服务器端接收　求并处理　求，然后把回应数据发　给客户端，客户端　取数据，最后关　　接，　次交互结束　　‥
- 基于UDP：UDP 协议是用户数据报协议的简称，也用于网络数据的传　　虽然 UDP 协议是　种不太可　的协议，但有时在　要　快地接收数据并且可以忍受　小　误的情况下，UDP 就会表现出更大的优势　我客户端只　要发　，服务端能不能接收的到我不管

4 TCP协　Socket代　　例：

`先运行服务端，在运行客户端`，

1. 服务端：

```
package com.test.io;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

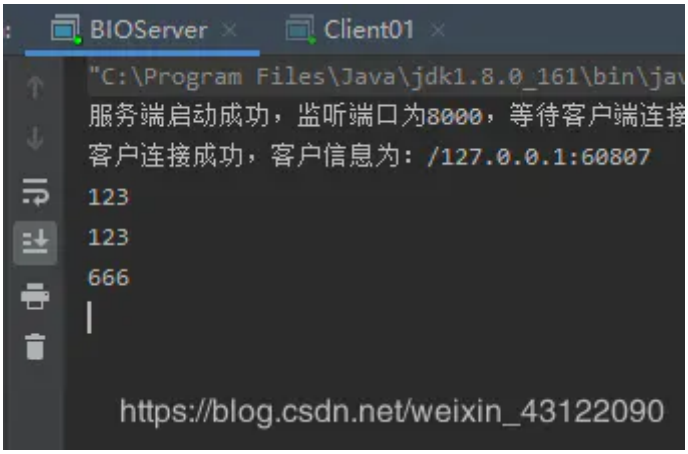
```
import java.net.ServerSocket;
import java.net.Socket;

//TCP协议Socket使用BIO进行通行：服务端
public class BIOServer {
    // 在main线程中执行下面这些代码
  public static void main(String[] args) {
        //1单线程服务
  ServerSocket server = null;
        Socket socket = null;
        InputStream in = null;
        OutputStream out = null;
        try {
            server = new ServerSocket(8000);
            System.out.println("服务端启动成功，监听端口为8000，等待客户端连接...");
            while (true){
                socket = server.accept(); //等待客户端连接
  System.out.println("客户连接成功，客户信息为：" +
socket.getRemoteSocketAddress());
                in = socket.getInputStream();
                byte[] buffer = new byte[1024];
                int len = 0;
                //读取客户端的数据
  while ((len = in.read(buffer)) > 0) {
                    System.out.println(new String(buffer, 0, len));
                }
                //向客户端写数据
  out = socket.getOutputStream();
                out.write("hello!".getBytes());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

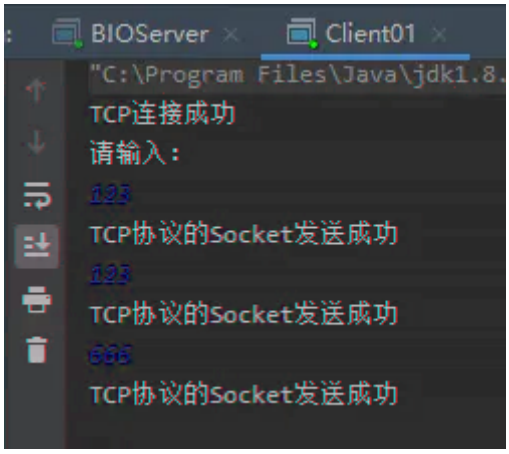
1. 客户端：

```
package com.test.io;

import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;

//TCP协议Socket：客户端
public class Client01 {
    public static void main(String[] args) throws IOException {
        //创建套接字对象socket并封装ip与port
        Socket socket = new Socket("127.0.0.1", 8000);
        //根据创建的socket对象获得一个输出流
  OutputStream outputStream = socket.getOutputStream();
        //控制台输入以IO的形式发送到服务器
  System.out.println("TCP连接成功 \n请输入：");
        while(true){
            byte[] car = new Scanner(System.in).nextLine().getBytes();
```

```
            outputStream.write(car);
            System.out.println("TCP协议的Socket发送成功");
            //刷新缓冲区
  outputStream.flush();
        }
    }
}
```

先运行服务端，在运行客户端 测试结果发 成功：





5 UDP协 Socket代 例：

先运行服务端，在运行客户端

1. 服务端：

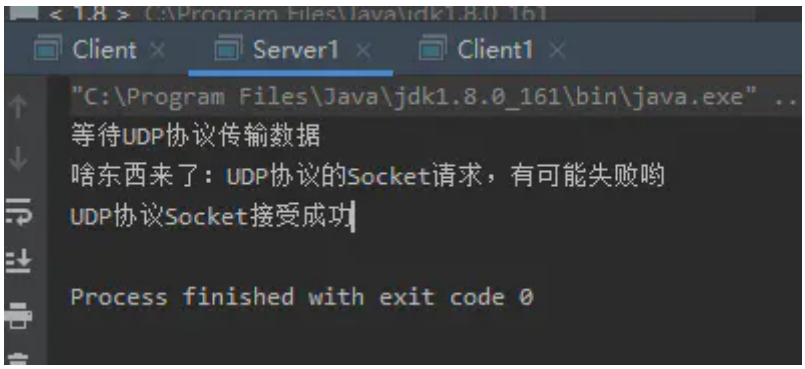
```
//UDP协议Socket：服务端
public class Server1 {
    public static void main(String[] args) {
        try {
            //DatagramSocket代表声明一个UDP协议的Socket
            DatagramSocket socket = new DatagramSocket(8888);
            //byte数组用于数据存储。
  byte[] car = new byte[1024];
            //DatagramPacket 类用来表示数据报包DatagramPacket
            DatagramPacket packet = new DatagramPacket(car, car.length);
            // //创建DatagramPacket的receive()方法来进行数据的接收,等待接收一个socket请
求后才执行后续操作;
  System.out.println("等待UDP协议传输数据");
            socket.receive(packet);
            //packet.getLength返回将要发送或者接收的数据的长度。
```

```
  int length = packet.getLength();
            System.out.println("啥东西来了：" + new String(car, 0, length));
            socket.close();
            System.out.println("UDP协议Socket接受成功");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

1. 客户端：

```
//UDP协议Socket：客户端
public class Client1 {
    public static void main(String[] args) {
        try {
            //DatagramSocket代表声明一个UDP协议的Socket
            DatagramSocket socket = new DatagramSocket(2468);
            //字符串存储人Byte数组
  byte[] car = "UDP协议的Socket请求，有可能失败哟".getBytes();
            //InetSocketAddress类主要作用是封装端口
  InetSocketAddress address = new InetSocketAddress("127.0.0.1", 8888);
            //DatagramPacket 类用来表示数据报包DatagramPacket
            DatagramPacket packet = new DatagramPacket(car, car.length,
address);
            //send() 方法发送数据包。
  socket.send(packet);
            System.out.println("UDP协议的Socket发送成功");
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

`先运行服务端，在运行客户端` 测试结果成功发 成功：



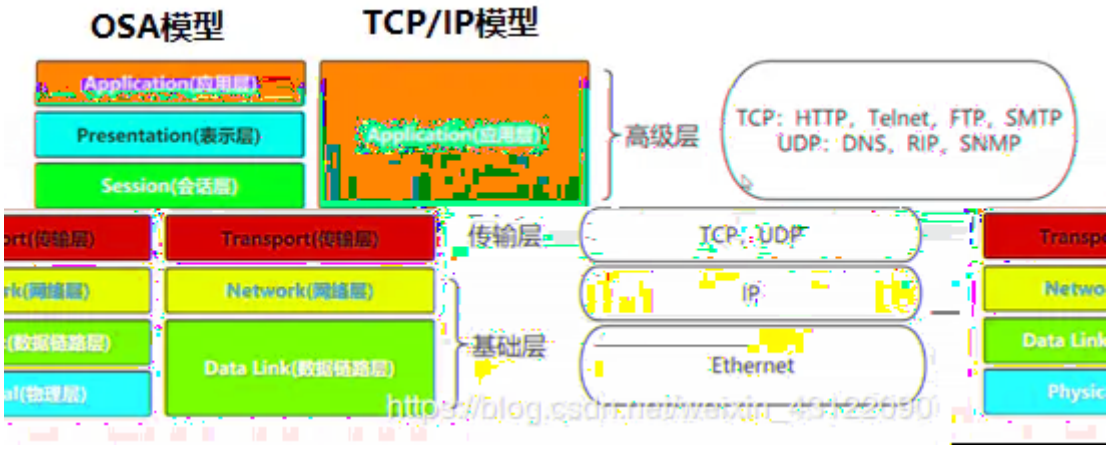
6 Socket 常

名	于	作
Socket	TCP 协议	Socket类同时工作于客户端和服务端，所有方法 是 用的， 个类三个主要作用，校 包信息，发 接（Client），操作流数据（Client/Server）
ServerSocket	TCP 协议	ServerSocket表示为服务端，主要作用就是绑定并监听 个服务器端口，为每个建立 接的客户端“克 /映射” 个Socket对 ，具体数据操作 是 个Socket对 完成的，ServerSocket只关注如何和客户端建立 接
DatagramSocket	ODP 协议	DatagramSocket 类用于表示发 和接收数据报包的套接字
DatagramPacket	ODP 协议	DatagramPacket 类用来表示数据报包，数据报包用来实现无 接包投 服务
InetAddress	IP+端口号	Java提供了InetAddress类来代表互联网协议（IP）地址，InetAddress类没有提供构 器，而是提供了如下两个 态方法来获取InetAddress实例：
InetSocketAddress	IP+端口号	在使用Socket来 接服务器时最简单的方式就是直接使用IP和端口，但Socket类中并未提供 种方式，而是SocketAddress的子类InetSocketAddress来实现 IP 地址 + 端口号的创建，不依 任何协议

3. HTTP

什么是Http协 ？

- Http协议是对客户端和服务器端之 数据之 实现可 性的传 文字 图片 视 等 文本数据的规范，格式简称为“ 文本传 协议”
- Http协议属于应用层，及用户访 的第 层就是http



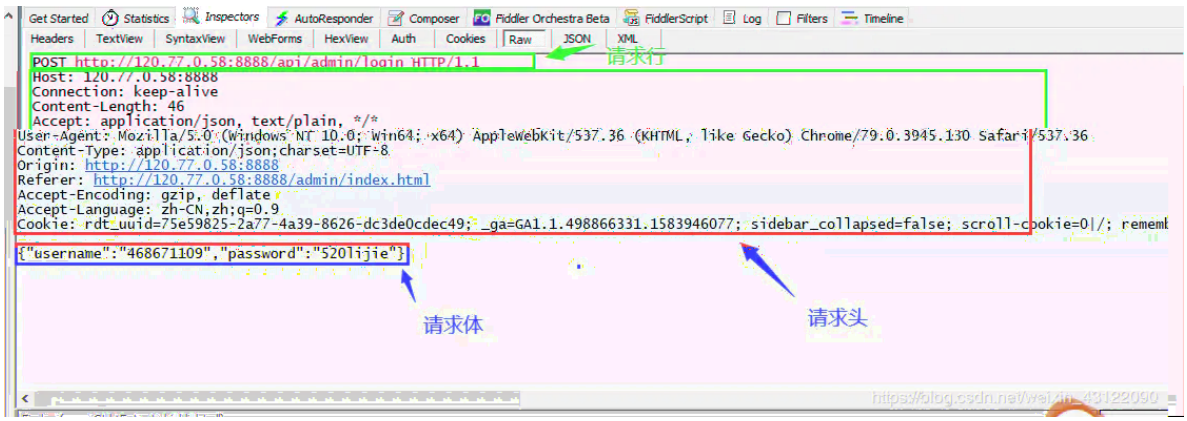
Socket和http　区别和应　场景

- Socket　接就是所　的　　接，理论上客户端和服务器端　旦建立　　接将不会主动断掉；
- Socket　用场景：网络游戏，　行持续交互，直播，在线视屏等
- http　接就是所　的短　接，即客户端向服务器端发　　次　求，服务器端响应后　接即会断开等待下次　接
- http　用场景：公司OA服务，互联网服务，电商，办公，网站等等等等

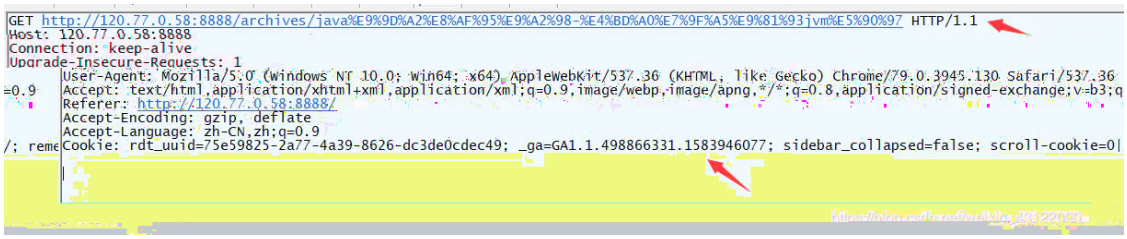
什么是http　　求体?

- HTTP　求体是我们　求数据时先发　给服务器的数据，毕竟我向服务器　数据，先要表明我要什么吧
- HTTP　求体由：　求行　　求头　　求数据组成的，
- 注意：GIT　求是没有　求体的

1. POST　求

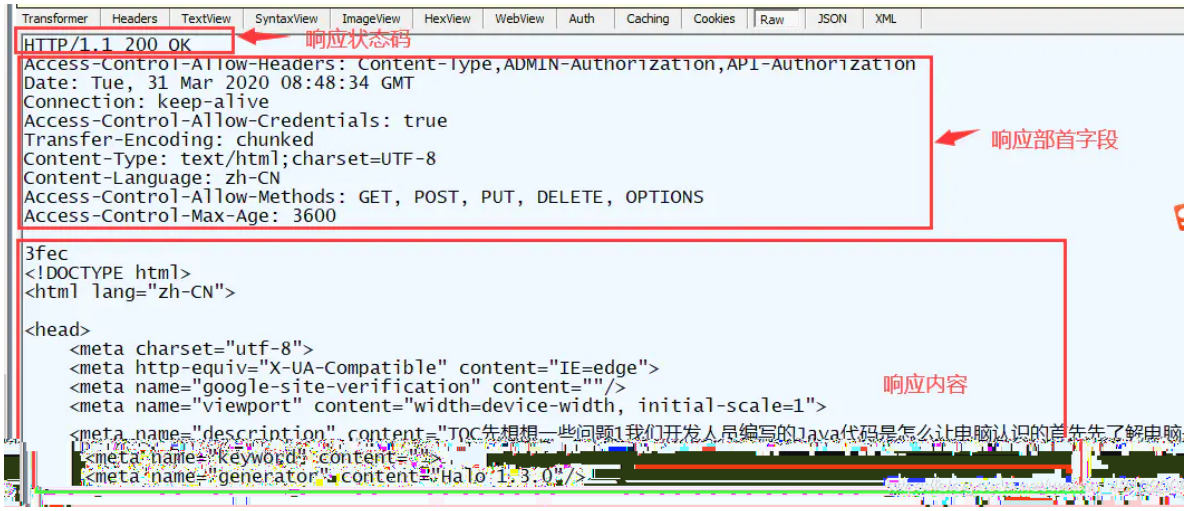


2. GIT　求是没有　求体的



http　响应报文有哪些?

- http的响应报是服务器　回给我们的数据，必　先有　求体再有响应报文
- 响应报文包含三　分 状态行　响应　　字段　响应内容实体实现



http和https　区别?

- 其实HTTPS就是从HTTP加上加密处理（　般是SSL安全　信线　）+认证+完整性保护
- 区别:
  1. http　要拿到ca证书，　要　的
  2. 端口不　样，http是80，https443
  3. http是　文本传　协议，信息是明文传　，https则是具有安全性的ssl加密传　协议
  4. http和https使用的是完全不同的　接方式（http的　接很简单，是无状态的；HTTPS 协议是由SSL+HTTP协议构建的可　行加密传　　份认证的网络协议，比http协议安全　）

HTTPS工作原理

- 　　先HTTP　求服务端生成证书，客户端对证书的有效期　合法性　域名是否与　求的域名一致　证书的公　（RSA加密）等　行校　；
- 二　客户端如果校　　后，就根据证书的公　的有效，生成　机数，　机数使用公　　行加密（RSA加密）；
- 三　消息体产生的后，对它的摘要　行MD5（或者SHA1）算法加密，此时就得到了RSA签名；
- 四　发　给服务端，此时只有服务端（RSA私　）能解密
- 五　解密得到的　机数，再用AES加密，作为密　（此时的密　只有客户端和服务端知　）

次完整　HTTP　求所　历几个步　?

HTTP通信机制是在一次完整的HTTP通信过程中，Web浏览器与Web服务器之间将完成下列7个步骤：

1. 建立TCP　接

怎么建立　接的，看上　的三次　手

2. Web浏览器向Web服务器发　　求行

旦建立了TCP　接，**Web浏　器就会向Web服务器发　　求命令**　例如：GET /sample/hello.jsp HTTP/1.1

3. Web浏览器发　　求头

浏览器发　其　求命令之后，　要以头信息的形式向Web服务器发　　些别的信息，**之后浏　器发了　　来　　服务器**，它已经结束了该头信息的发

4. Web服务器应答

客户机向服务器发出　求后，服务器会客户机回　应答，**HTTP/1.1 200 OK，应　　分是协　版本号和应　状**

5. Web服务器发　应答头

正如客户端会 同 求发 关于自 的信息 样，服务器也会 同应答向用户发 关于它自己的数据及被 求的文档

6. Web服务器向浏览器发 数据

Web服务器向浏览器发 头信息后，它会发 个空白行来表示头信息的发 到此为结束，接着，**它就以Content-Type应 头信 所描 格式发 户所 求 实 数据**

7. Web服务器关 TCP 接

常 HTTP状 是 么分 ，有哪些常 状 ？

- HTTP状态码表示客户端HTTP 求的 回结果 标识服务器处理是否正常 表明 求出现的 误等
- 状态码的类别：

> | 别 | 描 |
> | --- | --- |
> | 1xx： | 指示信息–表示 求已接收，正在处理 |
> | 2xx： | 成功–表示 求已被成功接收 理解 接受 |
> | 3xx： | 定向–要完成 求必 行更 步的操作 |
> | 4xx： | 客户端 误– 求有语法 误或 求无法实现 |
> | 5xx： | 服务器端 误–服务器未能实现合法的 求 |

- 常见的状态码：

状	描
200:	求被正常处理
204:	求被受理但没有 源可以 回
206:	客户端只是 求 源的 分，服务器只对 求的 分 源执行GET方法，相应报文中 Content-Range指定范围的 源
301:	永久性 定向
302:	临时 定向
303:	与302状态码有相似功能，只是它希望客户端在 求 个URI的时候，能 GET方法 定向到另 个URI上
304:	发 带条件的 求时，条件不满 时 回，与 定向无关
307:	临时 定向，与302类似，只是强制要求使用POST方法
400:	求报文语法有误，服务器无法识别
401:	求 要认证
403:	求的对应 源禁止被访
404:	服务器无法找到对应 源
500:	服务器内 误
503:	服务器正忙

Http协 中有 些 求方式

求方式	描
GET:	用于 求访 已经被URI（统 源标识符）识别的 源，可以 URL传参给服务器
POST:	用于传 信息给服务器，主要功能与GET方法类似，但 般推荐使用POST方式
PUT:	传 文件，报文主体中包含文件内容，保存到对应URI位置
HEAD:	获得报文 ，与GET方法类似，只是不 回报文主体， 般用于 证URI是否有 > 效
PATCH:	客户端向服务器传 的数据取代指定的文档的内容(分取代)
TRACE:	回显客户端 求服务器的原始 求报文，用于"回环"诊断
DELETE:	删 文件，与PUT方法相反，删 对应URI位置的文件
OPTIONS:	查询相应URI支持的HTTP方法

GET方法与POST方法 区别

- **区别** ： get 点在从服务器上获取 源，post 点在向服务器发 数据；

- **区别二**：Get传 的数据 小，因为受URL 度 制，但效率 ； Post可以传 大 数据，所以上传文件时只能用Post方式；
- **区别三**：get是不安全的，因为get 求发 数据是在URL上，是可见的，可能会泄 私密信息，如密码等； post是放在 求头 的，是安全的

http版本 对比

- HTTP1.0版本的特性：
  - 早先1.0的HTTP版本，是 种无状态 无 接的应用层协议
  - HTTP1.0规定浏览器和服务器保持短暂的 接，浏览器的每次 求 要与服务器建立 个TCP 接，服务器处理完成后立即断开TCP 接（无 接），服务器不 每个客户端也不记录 去的 求（无状态）
- HTTP1.1版本新特性
  - 认持久 接节省 信 ，只要客户端服务端任意 端没有明确提出断开TCP 接，就 直保持 接，可以发 多次HTTP 求
  - 管线化，客户端可以同时发出多个HTTP 求，而不用 个个等待响应
  - 断点续传原理
- HTTP2.0版本的特性
  - 二 制分帧（ 用二 制格式的编码将其封装）
  - 压缩（设置了专 的 压缩设计的HPACK算法 ）
  - 流 控制（设置了接收某个数据流的多少字节 些流 控制）
  - 多 复用（可以在共享TCP 接的基础上同时发 求和响应）
  - 求优先级（可以 优化 些帧的交 和传 序 步优化性能）
  - 服务器推 （就是服务器可以对 个客户端 求发 多个响应 服务器向客户端推 源无客户端明确的 求 （ 大更新））

什么是对 加密与 对 加密

- 对称密 加密是指加密和解密使用同 个密 的方式， 种方式存在的最大 就是密 发 ，即如何安全地将密 发给对方；
- 而 对称加密是指使用 对 对称密 ，即公 和私 ，公 可以 意发布，但私 只有自己知 发 密文的 方使用对方的公 行加密处理，对方接收到加密信息后，使用自己的私 行解密 由于 对称加密的方式不 要发 用来解密的私 ，所以可以保证安全性；但是和对称加密比 来， 常的慢

cookie和session对于HTTP有什么 ？

- HTTP协议本 是无法判断用户 份 所以 要cookie或者session

什么是cookie

- cookie是由Web服务器保存在用户浏览器上的文件（key-value格式），可以包含用户相关的信息 客户端向服务器发 求，就提取浏览器中的用户信息由http发 给服务器

什么是session

- session 是浏览器和服务器会话 程中，服务器会分 的 块储存空 给session
- 服务器 认为客户浏览器的cookie中设置 sessionid， 个sessionid就和cookie对应，浏览器在向服务器 求 程中传 的cookie 包含 sessionid ，服务器根据传 cookie 中的 sessionid 获取出会话中存储的信息，然后确定会话的 份信息

cookie与session区别

1. cookie数据存放在客户端上，安全性 差，session数据放在服务器上，安全性相对更

2. 单个cookie保存的数据不能　　4K，session无此　制 信息后，使用自己的私　　行解密　由于　对称加密的方式不　要发　用来解密的私　，所以可以保证安全性；但是和对称加密比　来，常的慢

cookie和session对于HTTP有什么　？

- HTTP协议本　是无法判断用户　份　所以　要cookie或者session

什么是cookie

- cookie是由Web服务器保存在用户浏览器上的文件（key-value格式），可以包含用户相关的信息　客户端向服务器发　　求，就提取浏览器中的用户信息由http发　给服务器

什么是session

- session 是浏览器和服务器会话　程中，服务器会分　的　块储存空　给session
- 服务器　认为客户浏览器的cookie中设置 sessionid，　个sessionid就和cookie对应，浏览器在向服务器　求　程中传　的cookie 包含 sessionid，服务器根据传　cookie 中的 sessionid 获取出会话中存储的信息，然后确定会话的　份信息

cookie与session区别

1. cookie数据存放在客户端上，安全性　差，session数据放在服务器上，安全性相对更
2. 单个cookie保存的数据不能　　4K，session无此　制
3. session　定时　内保存在服务器上，当访　增多，占用服务器性能，考虑到服务器性能方　，应当使用cookie