一篇文章搞定【所有】计算机网络八股文

陈同学在搬砖 2021-12-12 12:26

基本上可以命中校招面试80%的面试题

后台回复【计算机网络八股文】获取pdf版本

后续会持续更新其他科目

计算机网络八股文

Q1 计算机网络的七层模型/五层模型/四层模型?

Q2 网络层有哪些协议?

Q3 讲讲网络层IP协议中的ip地址?

介绍

给联网设备分配的一个在全世界范围是唯一的 32 位的标识符。

意义

- IP 地址是一种分等级的地址结构。分两个等级的好处是:
- 第一，IP 地址管理机构在分配 IP 地址时只分配网络号，而剩下的主机号则由得到该网络号的单位自行分配。这样就方便了 IP 地址的管理
- 第二，路由器仅根据目的主机所连接的网络号来转发分组（而不考虑目的主机号），这样就可以使路由表中的项目数大幅度减少，从而减小了路由表所占的存储空间。

编址方法

- 分类地址
  - 地址范围
  - 特殊ip
  - 私有ip
  - 类型
- 子网划分
  -
- 构造超网
  - 所要解决的问题是帮助减缓IP地址和路由表增大问题 采用CIDR的基本思想是取消IP地址的分类结构， 将多个地址块聚合在一起生成一个更大的网络，以包含更多的主机。
  - 在使用CIDR时，由于采用了网络前缀这种记法，IP地址由网络前缀和主机号这两部分组成。所以路由项目就要有网络前缀和下一跳地址。
  - 查找路由表时可能得到不止一个匹配结果，我们应当从匹配结果中选择具有最长网络前缀的路由。这叫做最长前缀匹配（longest-prefix matching）。这

是因为网络前缀越长，其地址块就越小，因而路由选择就越具体。

Q4 讲讲网络层IP协议中的ip数据报?

- 版本：ipv4/ipv6
- 首部长度
- 区分服务：说明数据包的紧急程度 主机端设置好紧急程度而且路由器 也得设置好紧急程度高的优先转发 这样才能工作
- 总长度：
- 标识：给数据包一个编号 每产生一个加1
- 标记:以太网的数据一般1500个字节有些大包路由器处理不了 就把它拆分成小包 这个标记就是说明这个包是否分片了 如下
- 片偏移: 如下占13 位，指出：较长的分组在分片后某片在原分组中的相对位置。片偏移以 8 个字节为偏移单位。
- 生存时间：生成数据包后会生成一个数TTL 会有一个初始值 每经过一次路由转发 TTL大小减一 这样可以防止数据包在路由之间的无限转发 比如数据包的目标ip并不存在 加入没有ttl则这个数据包会在路由之间无限转发 浪费带宽。记为 TTL (Time To Live)，指示数据报在网络中可通过的路由器数的最大值
- 协议:表明数据包的上层协议是什么（代表icmp/igmp/tcp等等的协议号）以便目的主机的 IP 层将数据部分上交给那个处理过程

Q5 讲讲网络层IP协议中的分组转发算法?

- 介绍
- 种类

Q6 讲讲网络层的虚拟专用网VPN?

介绍

实现

Q7 讲讲网络层的网络地址转换NAT?

介绍

工作原理

Q8 讲讲网络层的地址解析协议ARP?

arp介绍

- arp协议是根据IP地址获取MAC地址的一个协议。 主机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，

arq欺骗

- 上述Mac获取的过程会被利用做arp欺骗，假设局域网中另一台主机c想截获主机a和b之间的数据，只要在主机a进行广播寻mac地址时 报上自己的Mac地址 这样每次a发数据包时 让数据包先经过c再到b 就可以实现截获

利用上述截获过程 可以实现一台主机对局域网内的其他主机的宽带控制 和上网开关控制

Q9 讲讲网络层的 逆地址解析协议RARP?

实现机制

- RARP：逆地址解析协议，作用与 ARP 相反，是将 MAC 地址解析为 IP 地址的协议。主要用于无盘工作 站引导时获取 IP 地址。
- (1) 源端发送一个本地的 RARP 广播包，在此广播包中声明自己的 MAC 地址，并且请求任何收到此请 求的 RARP 服务器分配一个 IP 地址。
- （2）本地网段上的 RARP 服务器收到此请求后，检查其 RARP 列表，查找该 MAC 地址对应的 IP 地址。 如果存在，RARP 服务器就给源端发送一个响应数据包，并将此 IP 地址提供给对方主机使用；如果不存在， RARP 服务器对此不做任何响应。
- （3）源端在收到从 RARP 服务器来的响应信息后，利用得到的 IP 地址进行通信；如果一直没有收到 R ARP 服务器的响应信息，则表示初始化失败。 广播发送 RARP 请求分组，单播发送 RARP 响应分组。

Q10 讲讲网络层的 网际控制报文协议ICMP?

介绍

用来向主机或者路由器报告差错用的

ICMP 询问报文

- ①回送请求和回答报文

测试目的站是否可达以及了解其有关状态。

- ②时间戳请求和回答报文

可用来进行时钟同步和测量时间。

ICMP 差错报文

接收方收到询问报文以后 返回差错报告报文

ICMP差错报文有五种，即：

- ①终点不到达：目的主机不能交付数据报
- ②源点抑制：主机由于堵塞而丢弃数据报时，就向源点发送源点抑制报文，使源点知道应当把数据报的发送速率放慢。
- ③时间超过：当路由器收到生存时间为零的数据报时，除丢弃该数据报外，还要向源点发送时间超过报文。
- ④参数问题：当主机收到的数据报的首部中有的字段的值不正确时，就丢弃该数据报，并向源点发送参数问题报文。
- ⑤改变路由（重定向）：路由器把改变路由器报文发送给主机，让主机知道下次应将数据报发送给另外的路由器（可通过更好的路由）。

应用

- PING (Packet InterNet Groper)

```
PING 用来测试两个主机之间的连通性。
PING 使用了 ICMP 回送请求与回送回答报文。
```

```
两台笔记本电脑连起来后 ping 不通，
你觉得有哪些问题造成的
1 IP 是否在同一网段（ip 地址配置有问题）
2 防火墙是否禁用了 PING 回复
3 网卡是否启用（tcp/ip 安装的不完整）
4 还有就是网线具体接触是否正常
```

- Traceroute 的应用举例

```
在 Windows 操作系统
中这个命令是 tracert。

用来跟踪一个分组
从源点到终点的路径。

它利用 IP 数据报中的 TTL 字段
和 ICMP 时间超过差错报告报文
实现对从源点到终点的路径的跟踪。
```

Q11 传输层有哪些协议

TCP UDP协议

Q12 端口指的是什么?

Q13 TCP和UDP协议的区别在哪里?

- TCP的优缺点
- UDP的优缺点
- 总结

Q14 讲讲TCP协议的报文格式?

源端口和目的端口字段

序号字段——TCP 连接中传送的数据流中的
每一个字节都编上一个序号。
序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

确认号字段——是期望收到对方的下一个报文段的数据的第一个字节的序号。

数据偏移它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远（即首部长度）。

保留字段——占 6 位，保留为今后使用，但目前应置为 0。

紧急 URG —— 当 URG [?] 1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。

确认 ACK —— 只有当 ACK [?] 1 时确认号字段才有效。当 ACK [?] 0 时，确认号无效。

推送 PSH (PuSH) —— 接收 TCP 收到 PSH = 1 的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满了后再向上交付。

复位 RST (ReSeT) —— 当 RST [?] 1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。

同步 SYN —— 同步 SYN = 1 表示这是一个连接请求或连接接受报文。

终止 FIN (FINish) —— 用来释放一个连接。FIN [?] 1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

窗口字段 ——用来让对方设置发送窗口的依据，单位为字节。

检验和 —— 检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

紧急指针字段 —— 占 16 位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。

选项字段 —— 长度可变。TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长

```
 MSS (Maximum Segment Size)

是 TCP 报文段中的数据字段的最大长度。 数据字段加上 TCP 首部才等于整个的 TCP 报文段。 所以，MSS是“TCP 报文段长度减去 TCP 首部长度”。
 为什么要规定 MSS ？MSS 与接收窗口值没有关系。 若选择较小的 MSS 长度，网络的利用率就降低。
 当 TCP 报文段只含有 1 字节的数据时，
在 IP 层传输的数据报的开销至少有 40 字节
(包括 TCP 报文段的首部和 IP 数据报的首部)。
这样，对网络的利用率就不会超过 1/41。
到了数据链路层还要加上一些开销。
 若 TCP 报文段非常长，那么在 IP 层传输时就有可能要分解成多个短数据报片。在终点要把收到的各个短数据报片装配成原来的 TCP 报文段。
当传输出错时还要进行重传。这些也都会使开销增大。

 因此，MSS 应尽可能大些，只要在 IP 层传输时不需要再分片就行。

 由于 IP 数据报所经历的路径是动态变化的，因此在这条路径上确定的不需要分片的 MSS，如果改走另一条路径就可能需要进行分片。

因此最佳的 MSS 是很难确定的。

 其他选项

窗口扩大选项 ——其中有一个字节表示移位值 S。新的窗口值等于 TCP 首部中的窗口位数增大到 (16 + S)，相当于把窗口值向左移动 S 位后获得实际

时间戳选项——其中最主要的字段时间戳值字段（4 字节）和时间戳回送回答字段（4 字节）。

选择确认选项——在后面的 5.6.3 节介绍。

填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。
```

Q15 讲讲TCP协议的可靠传输?

介绍

理想传输条件有： ①传输信道不产生差错。

②不管发送方以多快的速度发送数据，

接收方总是来得及处理收到的数据。 一句话就是说:不丢包,不重复,次序不乱,数据不错.

当出现差错时让发送方重传出现差错的数据，所以引入了可靠传输

同时在接收方来不及处理收到的数据时，及时告诉发送方适当降低发送数据的速度。所以引入了流量控制 拥塞控制.

滑动窗口机制

-
-
-
-

停止等待ARQ

- 介绍
- 过程

- 特点

连续 ARQ 协议

- 介绍
- 退后N帧确认方式
- 选择确认SACK

Q16 讲讲TCP协议的流量控制

介绍

```
流量控制 (flow control)
就是让发送方的发送速率不要太快,
防止接收方接受窗口溢出产生丢包
既要让接收方来得及接收,
也不要使网络发生拥塞。
利用滑动窗口机制可以很方便地
在 TCP 连接上实现流量控制。
```

控制方式

死锁问题

效率问题

`接收方糊涂窗口综合症`

- 出现问题

当接收方的 TCP 缓冲区已满，接收方会向发送方发送窗口大小为 0 的报文。

若此时接收方的应用进程以交互方式每次只读取一个字节，于是接收方又发送窗口大小为一个字节的更新报文，发送方应邀发送一个字节的数据（发送的 IP 数据报是 41 字节长），于是接收窗口又满了，如此循环往复。

- 解决方法

让接收方等待一段时间，使得或者接收缓存已有足够空间容纳一个最长的报文段，或者等到接收缓存已有一半空闲的

空间。只要出现这两种情况之一，接收方就发出确认报文，并向发送方通知当前的窗口大小。

`发送方糊涂窗口综合症`

- 出现问题

发送方 TCP 每次接收到一字节的数据后就发送。

这样，发送一个字节需要形成 41 字节长的 IP 数据报。若接收方确认，并回送这一字节，就需传送总长度为 162 字节共 4 个报文段。效率很低。

- Nagle算法

若发送应用进程把要发送的数据逐个字节地送到 TCP 的发送缓存，

则发送方就把第一个数据字节先发送出去，把后面到达的数据字节都缓存起来。

当发送方收到对第一个数据字符的确认后，

再把发送缓存中的所有数据组装成一个报文段发送出去，同时继续对随后到达的数据进行缓存。

只有在收到对前一个报文段的确认后才继续发送下一个报文段。

当到达的数据已达到发送窗口大小的一半或已达到报文段的最大长度时，就立即发送一个报文段。

Q17 讲讲TCP协议的拥塞控制?

介绍

和流量控制的区别

拥塞控制

防止过多的数据注入到网络中，
这样可以使网络中的路由器
或链路不致过载。
拥塞控制前提是网络
能够承受现有的网络负荷。

流量控制

指点对点通信量的控制，
是个端到端的问题
（接收端控制发送端）。
流量控制就是
要抑制发送端发送数据的速率，
以便使接收端来得及接收。

相似之处

--某些拥塞控制算法
是向发送端发送控制报文，
并告诉发送端，
网络已出现麻烦，
必须放慢发送速率。
这点和流量控制是相似的。

原则

判断

解决办法

慢开始

拥塞避免

快重传

快恢复

流程图

Q18 讲讲TCP协议的连接管理?

三次握手

- step1:第一次握手 建立连接时，客户端发送TCP连接请求数据报文到服务

器，（其中，SYN=1，ACK=0，表示这是一个TCP连接请求数据报文；序号seq=x，表明传输数据时的第一个数据字节的序号是x）。并进入SYN_SENT状态，等待服务器确认。

- step2:第二次握手 服务器收到请求后，自己也发送TCP连接响应数据报文（其中确认报文段中，标识位SYN=1，ACK=1，表示这是一个TCP连接响应数据报文，并含服务端的初始序号seq(服务器)=y，以及服务器对客户端初始序号的确认号ack(服务器)=seq(客户端)+1=x+1）。,确认客户的数据包=此时服务器进入SYN_RECV状态。

- step3:第三次握手 客户端收到服务器的连接响应报文，(标识位ACK=1向服务器发送一个序列号(seq=x+1)，确认号为ack(客户端)=y+1，)此包发送完毕，客户端和服务器进入ESTAB_LISHED(TCP连接成功)状态，完成三次握手。

- 未连接队列 在三次握手协议中，服务器维护一个未连接队列，该队列为每个客户端的SYN包(syn=j)开设一个条目，该条目表明服务器已收到SYN包，并向客户发出确认，正在等待客户的确认包时，删除该条目，服务器进入ESTAB_LISHED状态。`

1.为什么需要三次握手，
两次不可以吗?
或者四次、五次可以吗?

我们来分析一种特殊情况，
假设客户端请求建立连接，
发给服务器SYN包等待服务器确认，

服务器收到确认后，
如果是两次握手，
假设服务器给客户端
在第二次握手时发送数据，

数据从服务器发出，
服务器认为连接已经建立，

但在发送数据的过程中数据丢失，客户端认为连接没有建立，
会进行重传。

假设每次发送的数据一直在丢失，客户端一直SYN，
服务器就会产生多个无效连接，
占用资源，
这个时候服务器可能会挂掉。

这个现象就是
我们听过的“SYN的洪水攻击”。

总结：第三次握手是为了防止：
如果客户端迟迟没有收到服务器返回确认报文，
这时会放弃连接，
重新启动一条连接请求，
但问题是：
服务器不知道客户端没有收到，
所以他会收到两个连接，
浪费连接开销。
如果每次都是这样，
就会浪费多个连接开销

2.最后一个ack丢失怎么办?

如果此时ACK在网络中丢失，
那么Server端该TCP连接的状态为SYN_RECV，
并且依次等待3秒、6秒、12秒后重新发送SYN+ACK包，
以便Client重新发送ACK包。
Server重发SYN+ACK包的次数，
可以通过设置/proc/sys/net/ipv4/tcp_synack_retries修改，
默认值为5。
如果重发指定次数后，
仍然未收到ACK应答，
那么一段时间后，
Server自动关闭这个连接。
但是Client认为
这个连接已经建立，
如果Client端
向Server写数据，
Server端将以RST包
(用于强制关闭tcp连接)响应，
方能感知到Server的错误。

四次挥手

tcp 四次挥手，由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。

这个原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接。

收到一个 FIN 只意味着这一方向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。

首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

第一次挥手: 机 1 向主机 2 发送 FIN 报文段,表示关闭数据传送,并主机 1 进入 FIN_WAIT_1 状态,表示 没有数据要传输了

第二次挥手：机 2 接受到 FIN 报文后进入 CLOSE_WAIT 状态(被动关闭),然后发送 ACK 确认,表示同意了 主机 1 的关闭请求，这个时候主机 1 进入到 FIN_WAIT_2 状态，这个状态是相对来讲稍微持久一点的。

第三次挥手：机 2 等待主机 1 发送完数据,发送 FIN 到主机 1 请求关闭,主机 2 进入 LAST_ACK 状态,

第四次挥手：机 1 收到主机 2 发送的 FIN 后,回复 ACK 确认到主机 2,主机 1 进入 TIME_WAIT 状态,主机 2 收到主机 1 的 ACK 后就关闭连接了,状态为 CLOSED,主机 1 等待 2MSL,仍然没有收到主机 2 的回复,说明 主机 2 已经正常关闭了,主机 1 关闭连接

MSL（Maximum Segment Lifetime）：报文最大生存时间，是任何报文段被丢弃前在网络内的最长时间。当主机 1 回复主机 2 的 FIN 后，等待(2-4 分钟)，即使两端的应用程序结束。

```
疑问：
1.为什么需要2MSL时间?

首先，MSL即Maximum Segment Lifetime，
就是最大报文生存时间，
是任何报文在网络上的
存在的最长时间，
超过这个时间报文将被丢弃。
《TCP/IP详解》中
是这样描述的：
MSL是任何报文段被丢弃前
在网络内的最长时间。
RFC 793中规定MSL为2分钟，
实际应用中常用的是30秒、1分钟、2分钟等。

TCP的TIME_WAIT需要等待2MSL，
当TCP的一端发起主动关闭，
三次挥手完成后发送第四次挥手的ACK包后就进入这个状态，
等待2MSL时间主要目的是：
防止最后一个ACK包
对方没有收到，
那么对方在超时后
将重发第三次握手的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
才可以继续使用。
当连接处于2MSL等待阶段时任何迟到的报文段都将被丢弃。

2.为什么是四次挥手，而不是三次或是五次、六次?

双方关闭连接
要经过双方都同意。所以，
首先是客服端给服务器发送FIN，要求关闭连接，
服务器收到后
```

会发送一个ACK进行确认。
服务器然后再发送一个FIN,
客户端发送ACK确认,
并进入TIME_WAIT状态。
等待2MSL后自动关闭。

3.KeepAlive机制?

设想：客户已主动
与服务器建立了TCP连接,
到后来客户端出现故障,
服务器以后
不再收到客户发来的数据。
因此，必须有措施使服务器
不再白白等待下去。
这就是保活计时器,
服务器每收到一次客户的数据,
就重新设置保活计时器,
发送一个探测报文段。
若10个探测报文段没有响应,
服务器就认为客户端出了故障,
接着就关闭这个连接。
keepalive，是在 TCP 中一个可以检测死连接的机制。
keepalive 原理很简单,
TCP 会在空闲了一定时间
后发送数据给对方:

1.如果主机可达,
对方就会响应 ACK 应答,
就认为是存活的。

2.如果可达,
但应用程序退出,
对方就发 FIN 应答,
发送 TCP 撤消连接。

3.如果可达,
但应用程序崩溃,
对方就发 RST 消息。

4.如果对方主机不响应ack,rst,
继续发送直到超时,
就撤消连接。
这个时间就是默认的二个小时。

Q19 讲讲TCP连接及断开过程中的状态转换?

Q20 应用层有哪些协议?

Q21 讲讲DNS协议?

概述

域名

域名服务器

域名解析过程

Q22 讲讲HTTP协议?

概念

在这里插入图片描述

每个万维网网点都有一个服务器进程，它不断地监听TCP的端口80， 以便发现是否有浏览器（即万维网客户）向它发出连接建立请求。

在浏览器和服务器之间的请求和响应的交互，必须按照规定的格式和遵循一定的规则， 从而使传输过程更加可靠,有效,高效.这些格式和规则就是超文本传送协议HTTP。

它是万维网上能够可靠地交换文件（包括文本、声音、图像等各种多媒体文件）的重要基础。

1.HTTP 是超文本传输协议，也就是HyperText Transfer Protocol。

2.HTTP 是一个在计算机世界里专门在「两点」之间「传输」文字、图片、音频、视频等「超文本」数据的「约定和规范」。

3.HTTP 不只是用于从互联网服务器传输超文本到本地浏览器的协议HTTP ,也可以是服务器和服务器之间的通信,

4.HTTP 协议是一个双向协议。

工作过程

我们在上网时，浏览器是请求方 A ，网站就是应答方 B 。双方约定用 HTTP 协议来通信， 于是浏览器把请求数据发送给网站，网站再把一些数据返回给浏览器， 最后由浏览器渲染在屏幕， 就可以看到图片、视频了。

Http1.0的主要特点

优点

1. 简单

HTTP 基本的报文格式就是 header + body，很简单.

2. 灵活和易于扩展

HTTP协议里的各类请求方法`状态码、头字段等都是可以自定义扩展的。 同时 HTTP 由于是工作在应用层（ OSI 第七层），则它下层可以随意变化。

3. 应用广泛和跨平台

缺点:

1. 无状态

服务器不会去记忆 HTTP 的状态，所以不需要额外的资源来记录状态信息

但是会导致它在完成有关联性的操作时会非常麻烦。

例如登录->添加购物车->下单->结算->支付，这系列操作都要知道用户的身份才行。 但服务器不知道这些请求是有关联的，每次都要问一遍身份信息。

无状态的问题解决方法有cookie/session/token 具体见下方讲解

2. 明文传输

明文意味着在传输过程中的信息，是可阅读的， 通过F12 控制台或 Wireshark 抓包都可以直接肉眼查看， 方便调试。 但是也导致了信息泄露

明文传输的问题在https协议中得到了解决

3. 不安全

HTTP 比较严重的缺点就是不安全：

通信使用明文（不加密），内容可能会被窃听。比如，账号信息容易泄漏，那你号没了。 不验证通信方的身份，因此有可能遭遇伪装。比如，访问假的淘宝、拼多多，那你钱没了。 无法证明报文的完整性，所以有可能已遭篡改。比如，网页上植入垃圾广告，视觉污染，眼没了。

HTTP 的安全问题，可以用 HTTPS 的方式解决，也就是通过引入 SSL/TLS 层，使得在安全上达到了极致。

4.无连接

HTTP/1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接， 服务器完成请求处理后立即断开TCP连接 每个TCP连接只能发送一个请求。发送数据完毕，连接就关闭 如果还要请求其他资源，就必须再新建一个连接。 我们知道TCP连接的建立需要三次握手，是很耗费时间的一个过程。所以，HTTP/1.0版本的性能比较差。

Http1.0的请求报文

在这里插入图片描述

1.请求行

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成， 它们用空格分隔。例如，GET /index.html HTTP/1.1。

HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

而常见的有如下几种：

1).GET

最常见的一种请求方式，当客户端要从服务器中读取文档时，当点击网页上的链接或者通过在浏览器的地址栏输入网址来浏览网页的，使用的都是GET方式。GET方法要求服务器将URL定位的资源放在响应报文的数据部分，回送给客户端。使用GET方法时，请求参数和对应的值附加在URL后面，利用一个问号（“?”）代表URL的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind,这样通过GET方式传递的数据直接表示在地址中，所以我们可以把请求结果以链接的形式发送给好友。以用google搜索domety为例，Request格式如下：

可以看到，GET方式的请求一般不包含”请求内容”部分，请求数据以地址的形式表现在请求行。地址链接如下：

http://www.google.cn/search?hl=zh-CN&source=hp &q=domety&aq=f&oq=地址中”?”之后的部分就是通过GET发送的请求数据，我们可以在地址栏中清楚的看到，各个数据之间用”&”符号隔开。显然，这种方式不适合传送私密数据。另外，由于不同的浏览器对地址的字符限制也有所不同，一般最多只能识别1024个字符，所以如果需要传送大量数据的时候，也不适合使用GET方式。

2).POST

对于上面提到的不适合使用GET方式的情况，可以考虑使用POST方式，因为使用POST方法可以允许客户端给服务器提供信息较多。POST方法将请求参 数封装在HTTP请求数据中，以名称/值的形式出现，可以传输大量数据，这样POST方式对传送的数据大小没有限制，而且也不会显示在URL中。还以上面 的搜索domety为例，如果使用POST方式的话，格式如下：

可以看到，POST方式请求行中不包含数据字符串，这些数据保存在”请求内容”部分，各数据之间也是使用”&”符号隔开。POST方式大多用于页 面的表单中。因为POST也能完成GET的功能，因此多数人在设计表单的时候一律都使用POST方式，其实这是一个误区。GET方式也有自己的特点和优 势，我们应该根据不同的情况来选择是使用GET还是使用POST。

3).HEAD

HEAD就像GET，只不过服务端接受到HEAD请求后只返回响应头，而不会发送响应内容。当我们只需要查看某个页面的状态的时候，使用HEAD是非常高效的，因为在传输的过程中省去了页面内容。

```
GET 与 POST
说一下 GET 和 POST 的区别?

1.Get 方法的含义是请求从服务器获取资源，
这个资源可以是静态的文本、页面、图片视频等。

比如，你打开我的文章，浏览器就会发送 GET 请求给服务器，
服务器就会返回文章的所有文字及资源。

而POST 方法则是相反操作，
它向 URI 指定的资源提交数据，
数据就放在报文的 body 里。

比如，你在我文章底部，
敲入了留言后点击「提交」（暗示你们留言），
浏览器就会执行一次 POST 请求，
把你的留言文字放进了报文 body 里，
然后拼接好 POST 请求头，
通过 TCP 协议发送给服务器。

2.GET 和 POST 方法都是安全和幂等的吗?
```

先说明下安全和幂等的概念：

在 HTTP 协议里，
所谓的「安全」是指请求方法
不会「破坏」服务器上的资源。

所谓的「幂等」，
意思是多次执行相同的操作，
结果都是「相同」的。

那么很明显 GET 方法就是安全且幂等的，
因为它是「只读」操作，
无论操作多少次，
服务器上的数据都是安全的，
且每次的结果都是相同的。

POST 因为是「新增或提交数据」的操作，
会修改服务器上的资源，
所以是不安全的，
且多次提交数据就会创建多个资源，
所以不是幂等的。

3.GET提交，请求的数据会附在URL之后。
POST提交：把提交的数据放置在是HTTP包的
包体<request-body>中。
上文示例中红色字体标明的
就是实际的传输数据
因此，GET提交的数据会在
地址栏中显示出来，
而POST提交，
地址栏不会改变。

4.传输数据的大小

首先声明,HTTP协议没有对传输的数据大小进行限制，
HTTP协议规范
也没有对URL长度进行限制。
而在实际开发中存在的限制主要有：
GET:特定浏览器和服务器
对URL长度有限制，
例如IE对URL长度的限制是2083字节(2K+35)。
对于其他浏览器，
如Netscape、FireFox等，
理论上没有长度限制，
其限制取决于操作系统的支持。
因此对于GET提交时，
传输数据就会受到URL长度的限制。
POST:由于不是通过URL传值，
理论上数据不受限。
但实际各个WEB服务器会规定
对post提交数据大小进行限制，
Apache、IIS6都有各自的配置。

5.安全性：

POST的安全性要比GET的安全性高。
注意：这里所说的安全性和
上面GET提到的“安全”不是同个概念。
上面“安全”的含义
仅仅是不作数据修改，
而这里安全的含义
是真正的Security的含义，
比如：通过GET提交数据，
用户名和密码将明文出现在URL上，
因为(1)登录页面有可能被浏览器缓存，
(2)其他人查看浏览器的历史纪录

```
(2)其他人查看浏览器的历史记录，
那么别人就
可以拿到你的账号和密码了。

6.GET和POST的本质区别是什么?

使用GET，form中的数据将编码到url中，
而使用POST的form中的数据
则在http协议的header中传输。
在使用上，当且仅当请求幂等
（字面意思是请求任意次返回同样的结果，
本质是请求本身不会改变服务器数据和状态）
时使用GET，
当请求会改变服务器数据或状态时
（更新数据，上传文件），
应该使用POST。
```

2.首部行

- Host

客户端发送请求时，用来指定服务器的域名。

- Host: 有了 Host 字段，就可以将请求发往「同一台」服务器上的不同网站。

- Content-Length 字段

服务器在返回数据时，会有 Content-Length 字段，表明本次回应的数据长度。

- Content-Length: 1000 如上面则是告诉浏览器，本次服务器回应的数据长度是 1000 个字节，后面的字节就属于下一个回应了。

- Connection 字段

Connection 字段最常用于客户端要求服务器使用 TCP 持久连接，以便其他请求复用。

- image HTTP/1.1 版本的默认连接都是持久连接，但为了兼容老版本的HTTP，需要指定 Connection 首部字段的值为 Keep-Alive。

- Connection: keep-alive 一个可以复用的 TCP 连接就建立了，直到客户端或服务器主动关闭连接。但是，这不是标准字段。

- Content-Type 字段

Content-Type 字段用于服务器回应时，告诉客户端，本次数据是什么格式。

Content-Type: text/html; charset=utf-8 上面的类型表明，发送的是网页，而且编码是UTF-8。

客户端请求的时候，可以使用 Accept 字段声明自己可以接受哪些数据格式。

- Accept: /上面代码中，客户端声明自己可以接受任何格式的数据。

- Content-Encoding 字段

Content-Encoding 字段说明数据的压缩方法。表示服务器返回的数据使用了什么压缩格式

- Content-Encoding: gzip 上面表示服务器返回的数据采用了 gzip 方式压缩，告知
  客户端需要用此方式解压。

客户端在请求时，用 Accept-Encoding 字段说明自己可以接受哪些压缩方法。

3.实体主体

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

Http1.0的响应报文

1.状态行

响应报文的开始行是状态行。状态行包括三项内容，即 HTTP 的版本，状态码，以及解释状态码的简单短语。 HTTP-Version Status-Code Reason-Phrase CRLF

其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase表示状态代码的文本描述。状态代码由三位数字组成，第一个数字定义了响应的类别，且有五种可能取值。

```
1xx

1xx 类状态码属于提示信息，是协议处理中的一种中间状态，实际用到的比较少。

2xx

2xx 类状态码表示服务器成功处理了客户端的请求，也是我们最愿意看到的状态。

「200 OK」是最常见的成功状态码，表示一切正常。如果是非 HEAD 请求，服务器返回的响应头都会有 body 数据。

「204 No Content」也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据。

「206 Partial Content」是应用于 HTTP 分块下载或断电续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理

3xx

3xx 类状态码表示客户端请求的资源发送了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是重定向。

「301 Moved Permanently」表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问。

「302 Moved Permanently」表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问。

301 和 302 都会在响应头里使用字段 Location，指明后续要跳转的 URL，浏览器会自动重定向新的 URL。

「304 Not Modified」不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，用于缓存控制。

4xx

4xx 类状态码表示客户端发送的报文有误，服务器无法处理，也就是错误码的含义。

「400 Bad Request」表示客户端请求的报文有错误，但只是个笼统的错误。

「403 Forbidden」表示服务器禁止访问资源，并不是客户端的请求出错。

「404 Not Found」表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端。

5xx

5xx 类状态码表示客户端请求报文正确，但是服务器处理时内部发生了错误，属于服务器端的错误码。

「500 Internal Server Error」与 400 类型，是个笼统通用的错误码，服务器发生了什么错误，我们并不知道。

「501 Not Implemented」表示客户端请求的功能还不支持，类似“即将开业，敬请期待”的意思。

「502 Bad Gateway」通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误。

「503 Service Unavailable」表示服务器当前很忙，暂时无法响应服务器，类似“网络服务正忙，请稍后重试”的意思。
```

2.首部行

响应头用于描述服务器的基本信息，以及数据的描述，服务器通过这些数据的描述信息， 可以通知客户端如何处理等一会儿它回送的数据。

3.实体主体

响应体就是响应的消息体，如果是纯数据就是返回纯数据，如果请求的是HTML页面， 那么返回的就是HTML代码，如果是JS就是JS代码，如此之类。

Http1.0的cookie/session/token

1.存在意义

如上面所说 http1.0是无状态的, 但是在有些时候是需要保存一些客户端的请求信息，识别客户端的某些状态，智能的、有针对性的去分析某些客户端的习惯。 这些时候，就需要去记录客户端的连接状态，识别请求的状态等。所以为了解决类似的事情，就需要使用到了 Cookie 和 Session。

2.cookie

Cookie：在客户端访问某个地址时，会将请求交到服务器进行处理，在发送请求的时候，浏览器会将页面的头部信息一并的交到服务器端进行处理。在处理的过程中，Cookie 在服务器端生成 ，在此同时，可以将一些需要保存的信息，存放到此 Cookie 中。生成 Cookie 对象时，需要确定具体的名称及具体的值，可以设置当前 Cookie 的过期时间，设置过期时间后，就相当于持久化了 Cookie 中的数据，此时的 Cookie 会以之前的 Cookie 名称，保存在客户端。 如果不设置过期时间，则当前 Cookie 的生命期是浏览器会话期间，一旦关闭了该浏览器，当前的Cookie 就会不存在了，此时的 Cookie 信息是保存在内存中。在服务器端，处理完后，会将生成的 Cookie ，随着 Http 响应，会在 Http 响应头中，加上Cookie 信息，浏览器接受到响应后，会按照 Http 响应头里的 Cookie ，在客户端建立 Cookie 。在下次客户进行请求的时候，Http 会附带着已经存储过的 Cookie，一并发送到服务器。一个域，在客户端建立的所以 Cookie 都是可以共享的，只要 Cookie 没有过期。

3.session

Session：Session 是在服务器端生成的，存储在服务器端，即存在内存中。可以对生成的 Session 设置过期时间， 如果不设置过期时间，默认的 Session 过期时间是30 分钟（在不同的服务器中，它的过期时间略有不同，本文是以 Tomcat 来说的）
但是，Sesssion 的生成的同时，会生成一个与之相关联的的 SessionID ，此 SessionID的存储是需要 Cookie 来完成的。SessionID 是以名称为 JSESSIONID，其值应该是一个既不会重复，又不容易被找到规律以仿造的字符串。SessionID会随着此次 Http 响应，一并返回到客户端，并保存在客户端中。 到当前请求再次发出后，该 SessionID会随着 Http 头部，传到服务器中， 服务器依据当前 SessionID 得到与之对应的 Session.

其中：通过 Cookie 的方式存储 Session 状态，只是其中一种方式。如果客户端禁用了 Cookie 的话， 很多网站任然可以存储用户的信息。一种处理的方式是URL 重写， 将 SesseionID 直接附加在请求地址的后面。另一种处理的方式是，使用隐藏自动的方式。就是服务器自动的在表单中，添加一个隐藏字段， 以便在表单提交时，将 SesseionID 一起传到服务器，进行识别。

4.token

Q23 讲讲HTTP1.1协议?

Http 1.1 的改进

1.缓存处理

在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag，If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。

2.带宽优化及网络连接的使用

HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，

并且不支持断点续传功能，HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

3.错误通知的管理

在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

4.Host头处理

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。

5. 长连接

在早期的HTTP/1.0中，每次http请求都要创建一个连接，而创建连接的过程需要消耗资源和时间，为了减少资源消耗，缩短响应时间，就需要重用连接。在后来的HTTP/1.0中以及HTTP/1.1中，引入了重用连接的机制，就是在http请求头中加入Connection: keep-alive来告诉对方这个请求响应完成后不要关闭，下一次咱们还用这个请求继续交流。协议规定HTTP/1.0如果想要保持长连接，需要在请求头中加上Connection: keep-alive，而HTTP/1.1默认是支持长连接的，有没有这个请求头都行。

当然了，协议是这样规定的，至于支不支持还得看服务器（比如tomcat）和客户端（比如浏览器）的具体实现。在实践过程中发现谷歌浏览器使用HTTP/1.1协议时请求头中总会带上Connection: keep-alive，另外通过httpclient使用HTTP/1.0协议去请求tomcat时，即使带上Connection: keep-alive请求头也保持不了长连接。如果HTTP/1.1版本的http请求报文不希望使用长连接，则要在请求头中加上Connection: close，接收到这个请求头的对端服务就会主动关闭连接。

但是http长连接会一直保持吗？肯定是不会的。一般服务端都会设置keep-alive超时时间。超过指定的时间间隔，服务端就会主动关闭连接。同时服务端还会设置一个参数叫最大请求数，比如当最大请求数是300时，只要请求次数超过300次，即使还没到超时时间，服务端也会主动关闭连接。

2.Transfer-Encoding和Content-Length 谈到http长连接，都绕不开这两个请求/响应头。其中Transfer-Encoding不建议在请求头中使用，因为无法知道服务端能否解析这个请求头，而应该在响应头中使用，因为客户端浏览器都能解析这个响应头。Content-Length在请求方法为GET的时候不能使用，在请求方法为POST的时候需要使用，同时也常常出现在响应头中。为了方便描述，下面只说明响应头中出现这两个属性的情况。

要实现长连接很简单，只要客户端和服务端都保持这个http长连接即可。但问题的关键在于保持长连接后，浏览器如何知道服务器已经响应完成？在使用短连接的时候，服务器完成响应后即关闭http连接，这样浏览器就能知道已接收到全部的响应，同时也关闭连接（TCP连接是双向的）。在使用长连接的时候，响应完成后服务器是不能关闭连接的，那么它就要在响应头中加上特殊标志告诉浏览器已响应完成。

一般情况下这个特殊标志就是Content-Length，来指明响应体的数据大小，比如Content-Length: 120表示响应体内容有120个字节，这样浏览器接收到120个字节的响应体后就知道了已经响应完成。

由于Content-Length字段必须真实反映响应体长度，但实际应用中，有些时候响应体长度并没那么好获得，例如响应体

来自于网络文件，或者由动态语言生成。这时候要想准确获取长度，只能先开一个足够大的内存空间，等内容全部生成好再计算。但这样做一方面需要更大的内存开销，另一方面也会让客户端等更久。这时候Transfer-Encoding: chunked响应头就派上用场了，该响应头表示响应体内容用的是分块传输，此时服务器可以将数据一块一块地分块响应给浏览器而不必一次性全部响应，待浏览器接收到全部分块后就表示响应结束。

6. 管道网络传输

HTTP/1.1 采用了长连接的方式，这使得管道（pipeline）网络传输成为了可能。

长连接有两种工作方式，即非流水线方式和流水线方式。

非流水线方式，是客户在收到前一个响应后才能发出下一个请求。因此，在TCP连接已建立后，客户每访问一次对象都要用去一个往返时间RTT。这比非持续连接的两倍RTT的开销节省了建立TCP连接所需的一个RTT时间。但非流水线方式还是有缺点的，因为服务器在发送完一个对象后，其TCP连接就处于空闲状态，浪费了服务器资源。

流水线方式，是客户在收到HTTP的响应报文之前就能够接着发送新的请求报文。于是一个接一个的请求报文到达服务器后，服务器就可持续发回响应报文。因此，使用流水线方式时，客户访问所有的对象只需花费一个RTT时间。流水线工作方式使TCP连接中的空闲时间减少，提高了下载文档效率

即可在同一个 TCP 连接里面，客户端可以发起多个请求，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以减少整体的响应时间。

举例来说，客户端需要请求两个资源。以前的做法是，在同一个TCP连接里面， 先发送 A 请求，然后等待服务器做出回应，收到后再发出 B 请求。管道机制则是允许浏览器同时发出 A 请求和 B 请求。

但是服务器还是按照顺序，先回应 A 请求，完成后再回应 B 请求。要是 前面的回应特别慢，后面就会有许多请求排队等着。这称为「队头堵塞」。

Q24讲讲HTTP2.0 协议?

Http2.0的改进

HTTP/2 是 HTTP 协议自 1999 年 HTTP 1.1 发布后的首个更新，主要基于 SPDY 协议。

1. 头部压缩

HTTP/2 会压缩头（Header）如果你同时发出多个请求，他们的头是一样的或是相似的，那么，协议会帮你消除重复的分。

这就是所谓的 HPACK 算法：在客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就提高速度了。

2. 二进制格式

HTTP/2 不再像 HTTP/1.1 里的纯文本形式的报文，而是全面采用了二进制格式。

头信息和数据体都是二进制，并且统称为帧（frame）：头信息帧和数据帧。

这样虽然对人不友好，但是对计算机非常友好，因为计算机只懂二进制，那么收到报文后，无需再将明文的报文转成二进制，而是直接解析二进制报文，这增加了数据传输的效率。

3. 数据流

HTTP/2 的数据包不是按顺序发送的，同一个连接里面连续的数据包，可能属于不同的回应。因此，必须要对数据包做标记，指出它属于哪个回应。

每个请求或回应的所有数据包，称为一个数据流（Stream）。

每个数据流都标记着一个独一无二的编号，其中规定客户端发出的数据流编号为奇数， 服务器发出的数据流编号为偶数

客户端还可以指定数据流的优先级。优先级高的请求，服务器就先响应该请求。

4. 多路复用

HTTP/2 是可以在一个连接中并发多个请求或回应，而不用按照顺序一一对应。

移除了 HTTP/1.1 中的串行请求，不需要排队等待，也就不会再出现「队头阻塞」问题，降低了延迟，大幅度提高了连接的利用率。

举例来说，在一个 TCP 连接里，服务器收到了客户端 A 和 B 的两个请求，如果发现 A 处理过程非常耗时，于是就回应 A 请求已经处理好的部分，接着回应 B 请求，完成后，再回应 A 请求剩下的部分。HTTP/2 为了解决HTTP/1.1中仍然存在的效率问题，HTTP/2 采用了多路复用。即在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序一一对应。能这样做有一个前提，就是HTTP/2进行了二进制分帧，即 HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame）,并对它们采用二进制格式的编码。

也就是说，老板可以同时下达多个命令，员工也可以收到了A请求和B请求，于是先回应A请求，结果发现处理过程非常耗时，于是就发送A请求已经处理好的部分， 接着回应B请求，完成后，再发送A请求剩下的部分。A请求的两部分响应在组合到一起发给老板。

而这个负责拆分、组装请求和二进制帧的一层就叫做二进制分帧层。 除此之外，还有一些其他的优化，比如做Header压缩、服务端推送等。

5. 服务器推送

服务端推送是一种在客户端请求之前发送数据的机制。当代网页使用了许多资源:HTML、样式表、脚本、图片等等。在HTTP/1.x中这些资源每一个都必须明确地请求。这可能是一个很慢的过程。浏览器从获取HTML开始，然后在它解析和评估页面的时候，增量地获取更多的资源。因为服务器必须等待浏览器做每一个请求，网络经常是空闲的和未充分使用的。

为了改善延迟，HTTP/2引入了server push，它允许服务端推送资源给浏览器，在浏览器明确地请求之前。一个服务器经常知道一个页面需要很多附加资源，在它响应浏览器第一个请求的时候，可以开始推送这些资源。这允许服务端去完全充分地利用一个可能空闲的网络，改善页面加载时间。

Q25 讲讲HTTP3.0 协议?

Http3.0的改进

HTTP/2 主要的问题在于：多个 HTTP 请求在复用一个 TCP 连接，下层的 TCP 协议是不知道有多少个 HTTP 请求的。

所以一旦发生了丢包现象，就会触发 TCP 的重传机制，这样在一个 TCP 连接中的所有的 HTTP 请求都必须等待这个丢了的包被重传回来。

HTTP/1.1 中的管道（ pipeline）传输中如果有一个请求阻塞了，那么队列后请求也统统被阻塞住了

HTTP/2 多请求复用一个TCP连接，一旦发生丢包，就会阻塞住所有的 HTTP 请求。

这都是基于 TCP 传输层的问题，所以 HTTP/3 把 HTTP 下层的 TCP 协议改成了 UDP!

UDP 发生是不管顺序，也不管丢包的，所以不会出现 HTTP/1.1 的队头阻塞 和 HTTP/2 的一个丢包全部重传问题。

大家都知道 UDP 是不可靠传输的，但基于 UDP 的 QUIC 协议 可以实现类似 TCP 的可靠性传输。

QUIC 有自己的一套机制可以保证传输的可靠性的。当某个流发生丢包时，只会阻塞这个流，其他流不会受到影响。

TL3 升级成了最新的 1.3 版本，头部压缩算法也升级成了 QPack。

HTTPS 要建立一个连接，要花费 6 次交互，先是建立三次握手，然后是 TLS/1.3 的三次握手。QUIC 直接把以往的 TCP 和 TLS/1.3 的 6 次交互合并成了 3 次，减少了交互次数。

所以， QUIC 是一个在 UDP 之上的伪 TCP + TLS + HTTP/2 的多路复用的协议。

QUIC 是新协议，对于很多网络设备，根本不知道什么是 QUIC，只会当做 UDP，这样会出现新的问题。所以 HTTP/3 现在普及的进度非常的缓慢，不知道未来 UDP 是否能够逆袭 TCP。

Q26 讲讲HTTPs协议?

改进

区别

过程