计算机网络

协议

计算机与网络设备要相互通信，双方就必须基于相同的方法。比如，如何探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定。不同的硬件、操作系统之间的通信，所有的这一切都需要一种规则。而我们就把这种规则称为协议。常用的网络是在TCP/IP协议族的基础上运作的。

OSI，TCP/IP，五层协议的体系结构，以及各层协议

OSI分层 （7层）： 物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

TCP/IP分层（4层）： 网络接口层、 网际层、运输层、 应用层。

五层协议 （5层）： 物理层、数据链路层、网络层、运输层、 应用层。

每一层的协议如下：

- 物理层：RJ45、CLOCK、IEEE802.3 （中继器，集线器，网关）
- 数据链路：PPP、FR、HDLC、VLAN、MAC （网桥，交换机）
- 网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP、 （路由器）
- 传输层：TCP、UDP、SPX
- 会话层：NFS、SQL、NETBIOS、RPC
- 表示层：JPEG、MPEG、ASII
- 应用层：FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS

应用层（报文）

应用层、表示层、会话层

定义的是应用进程间的通信和交互的规则。

http、FTP（文件传输）、DNS（域名系统） 应用层（application-layer）的任务是通过应用进程间的交互来完成特定网络应用。

传输层 segment

直接为用户的应用进程提供服务。

由于一个主机可同时运行多个进程，因此运输层具有复用和分用的功能。复用是指多个应用层进程可同时使用运输层的服务，分用是指运输层把收到的信息分别交付给应用层中的相应进程。

网络层 packet

处理网络上流通的数据包，对数据包进行路由选择。

选择合适的网间路由和交换结点。在发送数据时，网络层把运输层产生的报文段或用户数据报封装成分组和包进行传送。

此外，网络层还可以实现拥塞控制、网际互连等功能。

链路层 frame

处理连接网络的硬件部分

两台主机之间的数据传输，总是在一段一段的链路上传送的，这就需要使用专门的链路层的协议。

在两个相邻节点之间传送数据时，数据链路层将网络层交下来的 IP 数据报组装成帧，在两个相邻节点间的链路上传送帧。每一帧包括数据和必要的控制信息（如：同步信息，地址信息，差错控制等）。

在接收数据时，控制信息使接收端能够知道一个帧从哪个比特开始和到哪个比特结束。这样，数据链路层在收到一个帧后，就可从中提出数据部分，上交给网络层。

控制信息还使接收端能够检测到所收到的帧中有无差错。如果发现差错，数据链路层就简单地丢弃这个出了差错的帧，以避免继续在网络中传送下去白白浪费网络资源。如果需要改正数据在链路层传输时出现差错（这就是说，数据链路层不仅要检错，而且还要纠错），那么就要采用可靠性传输协议来纠正出现的差错。这种方法会使链路层的协议复杂些。

物理层

在物理层上所传送的数据单位是比特。物理层的作用是实现相邻计算机节点之间比特流的透明传送，屏蔽掉具体传输介质和物理设备的差异。

使其上面的数据链路层不必考虑网络的具体传输介质是什么。“透明传送比特流”表示经实际电路传送后的比特流没有发生变化，对传送的比特流来说，这个电路好像是看不见的。

TCP各层作用

1. 物理层： 物理层规定了激活、维持、关闭通信端点之间的机械特性、电气特性、功能特性以及过程特性。在这一层，数据的单位称为比特。
2. 数据链路层： 数据链路层在不可靠的物理介质上提供可靠的传输。该层的作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。在这一层，数据的单位称为帧。
3. 网络层： 网络层负责对子网间的数据包进行路由选择。网络层还可以实现拥塞控制、网际互连等功能。在这一层，数据的单位称为数据包。

4. 传输层： 传输层是第一个端到端，即主机到主机的层次。传输层负责将上层数据分段并提供端到端的、可靠的或不可靠的传输。此外，传输层还要处理端到端的差错控制和流量控制问题。
5. 会话层： 会话层管理主机之间的会话进程，即负责建立、管理、终止进程之间的会话。会话层还利用在数据中插入校验点来实现数据的同步。
6. 表示层： 表示层对上层数据或信息进行变换以保证一个主机应用层信息可以被另一个主机的应用程序理解。表示层的数据转换包括数据的加密、压缩、格式转换等。
7. 应用层： 应用层为操作系统或网络应用程序提供访问网络服务的接口。

各层协议

- 应用层
  - 超文本传输协议HTTP
  - 文件传输协议FTP
  - 简单邮件传输协议SMTP
  - 域名系统DNS
  - 安全外壳协议SSH
  - 动态主机配置协议TELNET
- 传输层
  - 传输控制协议TCP
  - 用户数据报文协议UDP
- 网络层
  - 网际协议IP
  - 地址转换协议ARP
  - 反向地址转换协议RARP
  - Internet控制报文协议ICMP
  - Internet组管理协议IGMP
  - 路由信息协议RIP
  - 分布式链路状协议OSPF
  - 边界网关协议BGP
- 数据链路层
  - 自动重传请求协议ARQ
  - 停止等待协议CSMA/CD
  - 点对点协议PPP
- 物理层

- 中继器
- 集线器
- 网线
- HUB

常见协议

DNS（解析过程）应用层

DNS（Domain Name System）服务提供域名到IP地址之间的解析服务。

1、在浏览器中输入www . qq .com 域名，操作系统会先检查自己**本地的hosts文件**是否有这个网址映射关系，如果有，就先调用这个IP地址映射，完成域名解析。

2、如果hosts里没有这个域名的映射，则**查找本地DNS解析器缓存**，是否有这个网址映射关系，如果有，直接返回，完成域名解析。

3、如果hosts与本地DNS解析器缓存都没有相应的网址映射关系，首先会找TCP/IP参数中设置的首选DNS服务器-**本地DNS服务器**，此服务器收到查询时，如果要查询的域名，

- **包含在本地区域文件中**，则返回解析结果给客户机，完成域名解析，此解析具有**权威性**。
- 该域名不由本地DNS服务器区域解析，但该服务器已**缓存了此网址映射关系**，则调用这个IP地址映射，完成域名解析，此解析不具有权威性。

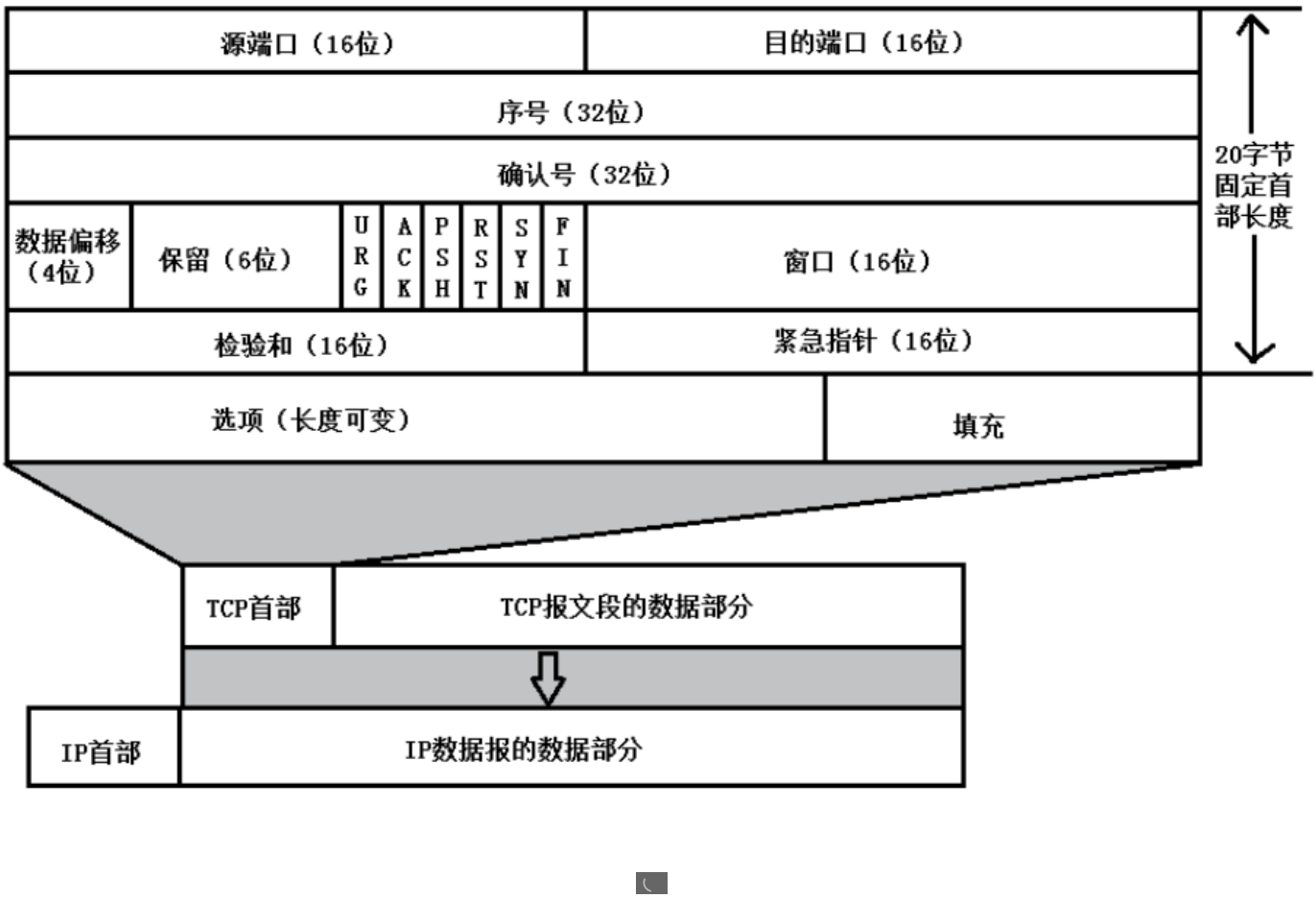
4、如果本地DNS服务器本地区域文件与缓存解析都失效，则根据本地DNS服务器的设置（是否设置转发器）进行查询

- 4.1、如果未用转发模式，本地DNS就把请求发至13台根DNS，**根DNS服务器**收到请求后会判断这个域名(.com)是谁来授权管理，并会返回一个负责该**顶级域名服务器的一个IP**。本地DNS服务器收到IP信息后，将会联系负责.com域的这台服务器。这台负责.com域的服务器收到请求后，如果自己无法解析，它就会找一个管理.com域的下一级DNS服务器地址(http://qq.com)给本地DNS服务器。当本地DNS服务器收到这个地址后，就会找http://qq.com域服务器，重复上面的动作，进行查询，直至找到www . qq .com主机。
- 4.2、如果用的是转发模式，此DNS服务器就会把请求**转发至上一级DNS服务器**，由上一级服务器进行解析，上一级服务器如果不能解析，或找根DNS或把转请求转至上上级，以此循环。不管是本地DNS服务器用是是转发，还是根提示，**最后都是把结果返回给本地DNS服务器**，由此DNS服务器再返回给客户机。

从客户端到本地DNS服务器是属于递归查询，而DNS服务器之间就是的交互查询就是迭代查询。

TCP协议

1. TCP（Transmission Control Protocol，传输控制协议）提供的是**面向连接，点对点的，全双工通信，可靠的，面向字节流**服务。
   - 字节流服务是指，为了方便传输，将大块数据分割成以**报文段（segment）**为单位的数据包进行管理。对于tcp来说，发送的是无结构的字节流。
     - 可靠的传输服务
       - 能够把数据准确可靠地传给对方。（依靠三次挥手四次握手）
       - 超时和重传机制（为了提高效率，使用了滑动窗口）
2. TCP首部格式



   - -seq包的序号，用来解决网络包乱序（reordering）问题。（seq的增加是和传输的字节数相关的）
   - -ack用于确认收到，用来解决不丢包的问题。 -
   - window滑动窗口（Sliding Window），用于解决流控的。
   - -tcp flag也就是包的类型，主要是用于操控TCP的状态机的。
3. 超时重传机制

TCP要保证所有的数据包都可以到达，所以，必需要有重传机制。接收端给发送端的Ack确认只会确认最后一个连续的包，SeqNum和Ack是以字节数为单位，所以ack的时候，不能跳着确认，只能确认最

大的连续收到的包，不然，发送端就以为之前的都收到了。

- 3.1 超时重传机制
  - 接收端一直不会ack，等待发送端等不到ack超时之后，会进行重传；1.只重传timeout的包 2.重传timeout后的所有数据
- 3.2 快速重传机制
  - 不以时间驱动，而以数据驱动重传。也就是说，如果，包没有连续到达，就ack最后那个可能被丢了的包，如果发送方连续收到3次相同的ack，就重传。Fast Retransmit的好处是不用等timeout了再重传。
- 3.3 数据确认流程
  - 首先客户端发送一个数据报，序号为1，数据长度是1000。这里有一个超时计时器。
  - 那么服务器接收到后，会发送一个确认报文，确认号为1000+1。表示你下一个该发送的是1001。
  - 那么如何处理发送丢包呢以及确认丢包呢
  - 当客户端的超时计时器到期前，没有收到服务器的确认，无论是发送的包丢了，还是确认的包丢了，都会重新发送数据报。
- 3.4 流量控制（滑动窗口）
  - 为什么要进行流量控制，是为了让发送方的发送速率不要太快，让接收方来得及接收，减少丢失。
  - 流量控制是作用于接收者的，它是控制发送者的发送速度从而使接收者来得及接收，防止分组丢失的。
  - 这里使用滑动窗口机制，例如在建立连接时，B告诉A接收窗口rwnd=400，那么发送方的窗口不能超过接收方给出的接收窗口数值。
  - 具体例子：
  - 建立连接后，A发送序号=1，数据1~100
  - A发送序号=101，数据101~200
  - A发送序号=201，数据201~300（丢失）
  - B发送确认ACK=1，确认号=201，rwnd=300,表示要从201开始发送到500截至
  - A发送序号=301，数据301~400
  - A发送序号=401，数据401~500
  - 201丢失，A发送序号=201，数据201~300
  - B发现太快了，发送确认ACK=1，确认号=501，rwnd=0，表示暂时不允许发送（零窗口）
  - 当过了一段时间后，有了一些空间，发送ACK=1，确认号=501，rwnd=300
  - 那么有一个问题了，当这个300的窗口报文丢失了怎么办呢？

- 这里有一个持续计时器，只要TCP一方收到对方的零窗口通知，就启动持续计时器，若持续时间到了，就给接收方发送一个，零窗口探测报文，若还是0，则重新定时，若不是0则打破死锁。
- 发送缓存
  - 用来暂时存放：
  - (1) 发送应用程序传送给发送方TCP准备发送的数据；
  - (2) TCP已发送出但尚未收到确认的数据
- 发送窗口后沿的后面部分表示已发送且已收到了确认。这些数据显然不需要再保留了。而发送窗口前沿的前面部分表示不允许发送的，因为接收方都没有为这部分数据保留临时存放的缓存空间。
- 发送窗口的位置由窗口前沿和后沿的位置共同确定。发送窗口后沿的变化情况有两种可能，即不动（没有收到新的确认）和前移（收到了新的确认）。发送窗口后沿不可能向后移动，因为不能撤销掉已收到的确认。
- 发送窗口前沿通常是不断向前移动，但也有可能不动。这对应于两种情况：一是没有收到新的确认，对方通知的窗口大小也不变；二是收到了新的确认但对方通知的窗口缩小了，使得发送窗口前沿正好不动。
- 接收缓存
  - 用来暂时存放：
  - (1) 按序到达的、但尚未被接收应用程序读取的数据；
  - (2) 未按序到达的数据。

- 3.5 拥塞控制
  - 拥塞避免和流量控制有什么区别呢?
  - 拥塞控制：拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况；常用的方法就是：（1）慢开始、拥塞避免（2）快重传、快恢复。（具体可以自己看看）
  - 为了更好对TCP进行拥塞控制，因特网建议标准定义了以下四种算法：慢开始，拥塞避免，快重传，快恢复。
  - 首先在TCP要求发送端维护两个窗口：
  - 1） 接收窗口rwnd，接收方根据当前缓存大小锁许诺的最新窗口值。
  - 2） 拥塞窗口cwnd ,发送方根据自己估算的网络拥塞程度而设置的窗口值。 发送窗口的上限是取这两者的最小值。
  - 慢开始： TCP刚连接好时，先令拥塞窗口cwnd =1 ,在每次收到一个对新报文段的确认时将cwnd加倍. Cwnd的大小呈指数增长。

- 拥塞避免算法： 当cwnd大于等于慢开始门限ssthresh时，cwnd窗口每次加1而不是加倍。当发送方检测到超时事件的发生时，就将慢开始门限设置为当前cwnd的一半，同时将cwnd设置为1. 这样的目的是迅速减少主机发送到网络的分组数，使得发生拥塞的路由器有足够的时间吧队列中积压的分组处理完毕。
- 快重传：当发送方连续收到三个重复的ACK报文时，直接重传对方尚未收到的报文段，而不必等待那个报文段设置的重传计时器超时。
- 快恢复：当发送端收到连续三个冗余的ACK时，就执行“乘法减小”算法，把慢开始门限ssthresh减半，cwnd设置为慢开始门限减半后的数值（与慢开始不同）。

TCP/UDP协议：传输层

1. TCP

是传输控制协议，它提供面向连接、可靠的字节流服务。当客户想要和服务器交换数据前，必须建立一个可靠的TCP连接，然后才能传输数据。TCP协议提供了超时重发、丢弃重复数据、检验数据、拥塞控制等功能，能够保证数据从一端传到另一端。

2. UDP

是用户数据报协议，它不是面向连接的，提供的是不可靠的服务。它只是运输层的一个简单传输协议，负责把应用程序传给IP层的数据报发送出去，但是不能保证到达目的地。由于UDP传输数据前不用在客户和服务器之间建立连接，也没有超时重发等机制，因此它的的传输速度比较快。

UDP（User Data Protocol，用户数据报协议）是一个简单的面向报文的运输层协议。它不提供可靠性，只是把应用程序传给IP层的数据报发送出去，但是不能保证它们能到达目的地。由于UDP在传输数据报前不用再客户和服务器之间建立一个连接，且没有超时重发等机制，所以传输速度很快。

3. TCP、UDP比较

TCP与UDP的区别

TCP	UDP
面向连接,需要三次握手，4次挥手	不需要建立连接
无差错，不丢失，不重复，且按序到达	不保证数据可靠、按序到达
面向字节流	面向报文
有拥塞控制,不会使源主机的发送速率降低	没有拥塞控制
连接只能是点到点	支持一对一，一对多，多对一和多对多的交互通信
首部开销20字节	首部开销小，只有8个字节
ftp telnet http https SMTP POP3	DNS DHCP tftp IGMP RTP

- TCP应用场景
  - 效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有UDP高。举几个例子：文件传输（准确高要求高、但是速度可以相对慢）、接受邮件、远程登录。
- UDP应用场景
  - 效率要求相对高，对准确性要求相对低的场景。举几个例子：QQ聊天、在线视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（广播、多播）。

IP协议

IP协议的作用是把各种数据包传送给对方。

IP间的通信依赖MAC地址。进行直接发送时会利用mac地址进行发送

实现两个基本功能：**寻址和分段**。

IP可以根据数据包包头中包括的目的地址将数据包传送到目的地址，在此过程中IP负责选择传送的道路，这种选择道路称为路由功能。如果有些网络内只能传送小数据包，IP可以将数据包重新组装并在数据报内注明。

1. IP地址
- IP 地址是指互联网协议地址，是 IP 协议提供的一种统一的地址格式，它为互联网上的每一台主机分配一个逻辑地址。

- 空间划分为 A、B、C、D、E 五类，其中 A、B、C 是基本类，D、E 类作为多播和保留使用，为特殊地址。
- 每个 IP 地址包括两个标识码（ID），即网络 ID 和主机 ID。同一个物理网络上的所有主机都使用同一个网络 ID，网络上的一个主机（包括网络上工作站，服务器和路由器等）有一个主机 ID 与其对应。

2. IP地址分类

- A 类地址：以 0 开头，第一个字节范围：0~127；
- B 类地址：以 10 开头，第一个字节范围：128~191；
- C 类地址：以 110 开头，第一个字节范围：192~223；
- D 类地址：以 1110 开头，第一个字节范围为 224~239；
- E 类地址：以 1111 开头，保留地址

3. 特殊的IP地址

- 255.255.255.255
  - 该IP地址指的是**受限的广播地址**。用于本地网络，路由器不会转发以受限广播地址为目的地址的分组；到了广播域的边界（网关）会自动终结
  - 一般广播地址既可在本地广播，也可跨网段广播。例如：主机192.168.1.1/30上的直接广播数据包后，另外一个网段192.168.1.5/30也能收到该数据报；若发送受限广播数据报，则不能收到。
- 0.0.0.0
  - 还没有获得ip地址的网卡
  - 常用于寻找自己的IP地址，若某个未知IP地址的无盘机想要知道自己的IP地址，它就以255.255.255.255为目的地址，向本地范围（局域网内）的服务器发送IP请求分组。
- 127.0.0.1
  - 本地环回地址，主要用于测试或网络管理/路由更新，比物理接口稳定。

ARP：通过ip寻址MAC地址

简单解释一些ARP地址解析协议的工作过程

广播发送ARP请求，单播发送ARP响应。

1. 首先，每个主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址之间的对应关系。
2. 当源主机要发送数据时，首先检查ARP列表中是否有对应IP地址的目的主机的MAC地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送ARP数据包，该数据包包括的内容有：源主机 IP地址，源主机MAC地址，目的主机的IP 地址。

3. 当本网络的所有主机收到该ARP数据包时，首先检查数据包中的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则首先从数据包中取出源主机的IP和MAC地址写入到ARP列表中，如果已经存在，则覆盖，然后将自己的MAC地址写入ARP响应包中，告诉源主机自己是它想要找的MAC地址。
4. 源主机收到ARP响应包后。将目的主机的IP和MAC地址写入ARP列表，并利用此信息发送数据。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

路由器工作原理——网络层

路由器是一种三层设备，是使用IP地址寻址：

1. 路由器接收到数据包，提取目标IP地址及子网掩码计算目标网络地址；
2. 根据目标网络地址查找路由表，如果找到目标网络地址就按照相应的出口发送到下一个路由器；
3. 如果没有找到，就看一下有没有默认路由，如果有就按照默认路由的出口发送给下一个路由器；
4. 如果没有找到就给源IP发送一个出错ICMP数据包表明没法传递该数据包；
5. 如果是直连路由就按照第二层MAC地址发送给目标站点。

交换机的实现原理——数据链路层

1. 交换机根据收到数据帧中的源MAC地址建立该地址同交换机端口的映射，并将其写入MAC地址表中。
2. 交换机将数据帧中的目的MAC地址同已建立的MAC地址表进行比较，以决定由哪个端口进行转发。
3. 如数据帧中的目的MAC地址不在MAC地址表中，则向所有端口转发。这一过程称为泛洪（flood）。
4. 广播帧和组播帧向所有的端口转发。

HTTP

URI和URL

与URI（统一资源标识符）相比，我们更熟悉URL（Uniform Resource Locator，统一资源定位符）。

URI用字符串标识某一互联网资源，而URL表示资源的地点（互联网上所处的位置）。可见URL是URI的子集。

HTTP无状态协议（cookie）

HTTP是一种不保存状态，即无状态（stateless）协议。

HTTP协议自身不对请求和响应之间的通信状态进行保存。这是为了更快地处理大量事务，确保协议的可伸缩性。

优点：由于不必保存状态，自然可减少服务器的CPU及内存资源的消耗。从另一侧面来说，也正是因为HTTP协议本身是非常简单的，所以才会被应用在各种场景里。

1. cookie

HTTP/1.1虽然是无状态协议，但为了实现期望的保持状态功能，于是引入了Cookie技术。

Cookie技术通过在请求和响应报文中写入Cookie信息来控制客户端的状态。

Cookie会根据从服务器端发送的响应报文内的一个叫做Set-Cookie的首部字段信息，通知客户端保存Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入Cookie值后发送出去。服务器端发现客户端发送过来的Cookie后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务器上的记录，最后得到之前的状态信息。

2. set-cookie

服务器准备开始管理客户端的状态时发送的

一次完整的HTTP请求过程

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

从http协议发送的客户端请求到达服务器端的整个传输过程。

1、客户端浏览器通过DNS解析到www.baidu.com的IP地址220.181.27.48，通过这个IP地址找到客户端到服务器的路径。客户端浏览器发起一个HTTP会话到220.161.27.48，然后通过TCP进行封装数据包，输入到网络层。

2、在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

3、客户端的网络层不用关心应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器。

4、客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

HTTP协议与TCP/IP协议的关系

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」。

HTTP的长连接和短连接本质上是TCP长连接和短连接。

HTTP属于应用层协议，在传输层使用TCP协议，在网络层使用IP协议。 IP协议主要解决网络路由和寻址问题，TCP协议主要解决如何在IP层之上可靠地传递数据包，使得网络上接收端收到发送端所发出的所有包，并且顺序与发送顺序一致。TCP协议是可靠的、面向连接的。

如何编写socket套接字

服务器端程序的编写步骤：

- 第一步：调用socket()函数创建一个用于通信的套接字。
- 第二步：给已经创建的套接字绑定一个端口号，这一般通过设置网络套接口地址和调用bind()函数来实现。
- 第三步：调用listen()函数使套接字成为一个监听套接字。
- 第四步：调用accept()函数来接受客户端的连接，这是就可以和客户端通信了。
- 第五步：处理客户端的连接请求。
- 第六步：终止连接。

客户端程序编写步骤：

- 第一步：调用socket()函数创建一个用于通信的套接字。
- 第二步：通过设置套接字地址结构，说明客户端与之通信的服务器的IP地址和端口号。
- 第三步：调用connect()函数来建立与服务器的连接。
- 第四步：调用读写函数发送或者接收数据。
- 第五步：终止连接。

session和cookie

1. cookie

- 位于用户的计算机上，用来维护用户计算机中的信息，直到用户删除。比如我们在网页上登录某个软件时输入用户名及密码时如果保存为cookie，则每次我们访问的时候就不需要登录网站了。我们可以在浏览器上保存任何文本，而且我们还可以随时随地的去阻止它或者删除。我们同样也可以禁用或者编辑cookie，但是有一点需要注意不要使用cookie来存储一些隐私数据，以防隐私泄

2. session

- session称为会话信息，位于web服务器上，主要负责访问者与网站之间的交互，当访问浏览器请求http地址时，将传递到web服务器上并与访问信息进行匹配， 当关闭网站时就表示会话已经结束，网站无法访问该信息了，所以它无法保存永久数据，我们无法访问以及禁用网站

3. session与cookie的区别

- （1）Cookie以文本文件格式存储在浏览器中，而session存储在服务端它存储了限制数据量。它只允许4kb它没有在cookie中保存多个变量。
- （2）cookie的存储限制了数据量，只允许4KB，而session是无限量的
- （3）我们可以轻松访问cookie值但是我们无法轻松访问会话值，因此它更安全
- （4）设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话。

4. 总结一下：

- Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；
- Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式。

持久连接（长连接）

为解决上述TCP连接的问题，HTTP/1.1和一部分的HTTP/1.0想出了持久连接的方法。持久连接的特点是，只要任意一端没有明确提出断开连接，则保持TCP连接状态。

http1.1是默认持久连接，如果要关闭就添加首部字段connect：close；1.1之前要开启则是Connect：keep-alive

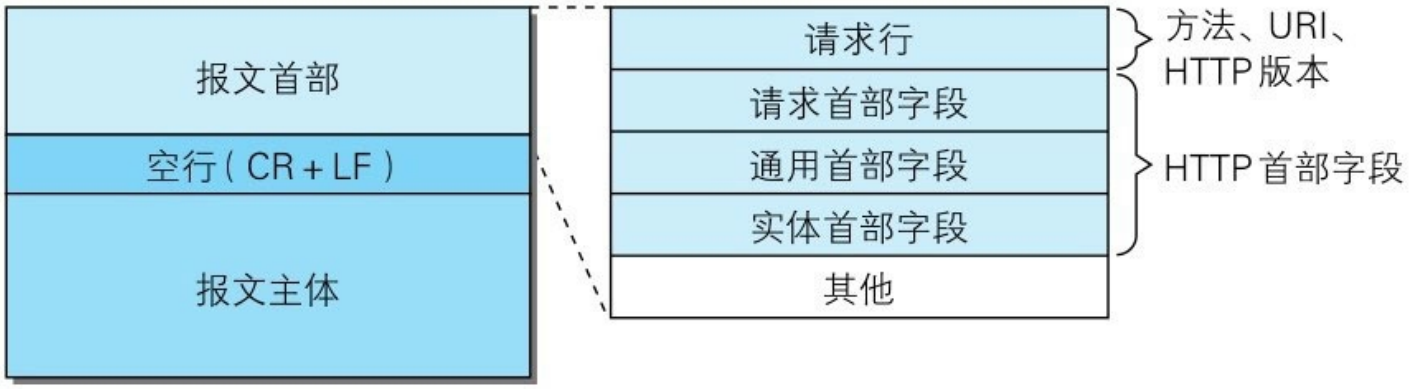
管线化

- 持久连接使得多数请求以管线化（pipelining）方式发送成为可能。
- 从前发送请求后需等待并收到响应，才能发送下一个请求。 管线化技术出现后，不用等待响应亦可直接发送下一个请求。这样就能够做到同时并行发送多个请求，而不需要一个接一个地等待响应了。
- 比如，当请求一个包含10张图片的HTML Web页面，与挨个连接相比，用持久连接可以让请求更快结束。而管线化技术则比持久连接还要快。请求数越多，时间差就越明显。

HTTP报文（组成）

http交互的信息被称为http报文，

HTTP请求报文由请求行、请求头、空行和请求内容4个部分构成。由**报文首部和报文主体**组成，由空行来划分。



报文首部：起至关重要的信息几乎都在这里

空行（CR+LF）

报文主体：所需要的用户和资源信息

报文首部

请求行/响应行

- POST /form/entry HTTP1.1
- 请求方法、URI、协议版本（3部分组成）
- HTTP/1.1 200 ok
- 协议版本、状态码、状态码原因短语（3部分组成）

请求首部字段/响应首部字段、通用首部字段、实体响应字段（共4种响应字段）

请求方法

- GET：获取资源；
- POST：传输实体主体；
- get和post两种请求的区别
  - 最主要的区别：get表示获取资源，post表示传输资源
  - Get和POST方法的区别
    - 1、传送方式：get通过地址栏传输，post通过报文传输。
    - 2、传送长度：get参数有长度限制（受限于url长度），而post无限制
    - 3、GET产生一个TCP数据包；POST产生两个TCP数据包。
    - 常见回答

找整理了一下，答案基本就是这么几条：

1：get在地址栏传输数据，post在表单传输数据；

2：get传输数据量小，post传输数据量大；

3：get传输数据可见，因此不安全；post传输数据不可见，因此安全；

    - 1、仅仅是浏览器的做法，不代表http协议
    - 2、http本身并没有做限定
    - 3、http为明文传输，并无安全性可言
- PUT：传输文件PUT方法用来传输文件（鉴于HTTP/1.1的PUT方法自身不带验证机制，任何人都可以上传文件，存在安全性问题，因此一般的Web网站不使用该方法）；
- DELETE：删除文件,是与PUT相反的方法。DELETE方法按请求URI删除指定的资源。不带验证机制，不使用。
- HEAD：获得报文首部,HEAD方法和GET方法一样，只是不返回报文主体部分。用于确认URI的有效性及资源更新的日期时间等。
- TRACE：**追踪路径**,TRACE方法是让Web服务器端将之前的请求通信环回给客户端的方法。
- OPTIONS：**询问支持的方法**,OPTIONS方法用来查询针对请求URI指定的资源支持的方法。
- CONNECT：要求用隧道协议连接代理;
    - CONNECT方法要求在与代理服务器通信时建立隧道，实现用隧道协议进行TCP通信。主要使用SSL和TLS协议把通信内容加密后经网络隧道传输。
- (connect,options在http1.1才开始使用)

响应码

- 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的响应结果表明请求被正常处理了**。
    - 200 OK 请求被正常处理
    - 204 No Content 请求被受理但没有资源可以返回
    - 206 Partial Content 该状态码表示客户端进行了范围请求，而服务器成功执行了这部分的GET请求。
- 3XX重定向 3XX响应结果表明浏览器需要执行某些特殊的处理以正确处理请求。
    - 301 Moved Permanently 永久性重定向。该状态码表示请求的资源已被分配了新的URI，以后应使用资源现在所指的URI
    - 302 Found 临时性重定向。该状态码表示请求的资源已被分配了新的URI，希望用户（本次）能使用新的URI访问。
    - 301 302状态码相似，但302状态码代表的资源不是被永久移动，只是临时性质的

    - 303 See Other 与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上
    - 303状态码和302 Found状态码有着相同的功能，但303状态码明确表示客户端应当采用GET方法获取资源，这点与302状态码有区别。当301、302、303响应状态码返回时，几乎所有的浏览器都会把POST改成GET，并删除请求报文内的主体，之后请求会自动再次发送。301、302标准是禁止将POST方法改变成GET方法的，但实际使用时大家都会这么做。
    - 304 Not Modified 该状态码表示客户端发送附带条件的请求时，服务器端允许请求访问资源，但因发生请求未满足条件的情况后，直接返回。304状态码返回时，不包含任何响应的主体部分。
    - 304虽然被划分在3XX类别中，但是和重定向没有关系。
    - 307 Temporary Redirect 临时重定向，与302类似，只是强制要求使用POST方法。尽管302标准禁止POST变换成GET，但实际使用时大家并不遵守。307会遵照浏览器标准，不会从POST变成GET
- 4XX客户端错误 4XX的响应结果表明客户端是发生错误的原因所在
    - 400：请求报文语法有误，服务器无法识别
    - 401 Unauthorized 请求需要认证
    - 403 Forbidden 请求的对应资源禁止被访问。服务器端没有必要给出拒绝的详细理由，但如果想作说明的话，可以在实体的主体部分对原因进行描述，这样就能让用户看到了
    - 404 Not Found 服务器无法找到对应资源。除此之外，也可以在服务器端拒绝请求且不想说明理由
- 5XX服务器错误 5XX的响应结果表明服务器本身发生错误
    - 500 Internal Server Error 服务器内部错误
    - 503 Service Unavailabl 服务器正忙。该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。如果事先得知解除以上状况需要的时间，最好写入Retry-After首部字段再返回给客户端

首部字段

无论是请求还是响应都会使用首部字段， 使用首部字段是为了给浏览器和服务器提供报文主体大小、所使用的语言、认证信息等内容。

- 字段类别
    - end-to-end端到端首部：分在此类别中的首部会转发给请求/响应对应的最终接收目标，且必须保存在由缓存生成的响应中，另外规定它必须被转发。不会被修改
    - hop-by-hop逐跳字段：分在此类别中的首部只对单次转发有效，会因通过缓存或代理而不再转发。HTTP/1.1和之后版本中，如果要使用hop-by-hop首部，需提供Connection首部字段。
- a、通用首部字段（请求报文与响应报文都会使用的首部字段）

    - Date：创建报文时间
    - Connection：连接的管理
    - Cache-Control：缓存的控制
    - Transfer-Encoding：报文主体的传输编码方式
- b、请求首部字段（请求报文会使用的首部字段）
    - Host：请求资源所在服务器
    - Accept：可处理的媒体类型
    - Accept-Charset：可接收的字符集
    - Accept-Encoding：可接受的内容编码
    - Accept-Language：可接受的自然语言
- c、响应首部字段（响应报文会使用的首部字段）
    - Accept-Ranges：可接受的字节范围
    - Location：令客户端重新定向到的URI
    - Server：HTTP服务器的安装信息
- d、实体首部字段（请求报文与响应报文的的实体部分使用的首部字段）
    - Allow：资源可支持的HTTP方法
    - Content-Type：实体主类的类型
    - Content-Encoding：实体主体适用的编码方式
    - Content-Language：实体主体的自然语言
    - Content-Length：实体主体的的字节数
    - Content-Range：实体主体的位置范围，一般用于发出部分请求时使用

----------手动分割线----------

- 通用首部字段
    - 请求报文和响应报文两方都会使用的首部。
    - Cache-Control：控制缓存的行为
        - 重用已获取的资源能够有效的提升网站与应用的性能。Web 缓存能够减少延迟与网络阻塞，进而减少显示某个资源所用的时间。
        - Cache-Control: no-store
            - 禁止进行缓存
            - 缓存中不得存储任何关于客户端请求和服务端响应的内容。每次由客户端发起的请求都会下载完整的响应内容。
        - Cache-Control: no-cache

        - 每次有请求发出时，缓存会将此请求发到服务器（译者注：该请求应该会带有与本地缓存相关的验证字段），服务器端会验证请求中所描述的缓存是否过期，若未过期（注：实际就是返回304），则缓存才使用本地缓存副本。
    - Connection：连接的管理
    - Date：创建报文的日期时间
    - Pragma：报文指令
    - Trailer：报文末端的首部一览
    - Transfer-Encoding：指定报文主体的传输编码方式
    - Upgrade：升级为其他协议
    - Via：代理服务器的相关信息
    - Warning：错误通知
- 请求首部字段
    - Accept 用户代理可处理的媒体类型
    - Accept-Charset 优先的字符集
    - Accept-Encoding　　优先的内容编码
    - Accept-Language　　优先的语言（自然语言）
    - Authorization Web 认证信息
    - Expect 期待服务器的特定行为
    - From 用户的电子邮箱地址
    - Host 请求资源所在服务器
    - If-Match　　比较实体标记
    - （ETag）If-Modified-Since 比较资源的更新时间
    - If-None-Match 比较实体标记（与 If-Match 相反）
    - If-Range 资源未更新时发送实体
    - Byte 的范围请求
    - If-Unmodified-Since　　比较资源的更新时间（与If-Modified-Since相反）
    - Max-Forwards 最大传输逐跳数
    - Proxy-Authorization　　代理服务器要求客户端的认证信息
    - Range 实体的字节范围请求
    - Referer　　对请求中
    - URI 的原始获取方
    - TE　　传输编码的优先级
    - User-Agent HTTP 客户端程序的信息

- 响应首部字段
  - Accept-Ranges 是否接受字节范围请求
  - Age 推算资源创建经过时间
  - ETag 资源的匹配信息
  - Location 令客户端重定向至指定
  - URIProxy-Authenticate 代理服务器对客户端的认证信息
  - Retry-After　　对再次发起请求的时机要求
  - Server　　HTTP服务器的安装信息
  - Vary 代理服务器缓存的管理信息
  - WWW-Authenticate 服务器对客户端的认证信息
- 实体首部字段
  - Allow　　资源可支持的HTTP方法
  - Content-Encoding　　实体主体适用的编码方式
  - Content-Language　　实体主体的自然语言
  - Content-Length 实体主体的大小（单位：字节）
  - Content-Location　　替代对应资源的
  - URIContent-MD5　　实体主体的报文摘要
  - Content-Range 实体主体的位置范围
  - Content-Type　　实体主体的媒体类型
  - Expires 实体主体过期的日期时间
  - Last-Modified　　资源的最后修改日期时间

报文主体和实体主体的差异

通常，报文主体等于实体主体。只有当**传输中进行编码操作时**，实体主体的内容发生变化，才导致它和报文主体产生差异。

- 报文

  是HTTP通信中的基本单位，由8位组字节流（octet sequence，其中octet为8个比特）组成，通过HTTP通信传输。

- 实体

  作为请求或响应的有效载荷数据（补充项）被传输，其内容由实体首部和实体主体组成。

- 内容编码

  内容编码指明应用在实体内容上的编码格式，并保持实体信息原样压缩。内容编码后的实体由客户端接收并负责解码。常用的内容编码有以下几种。

- ●gzip（GNU zip）●compress（UNIX系统的标准压缩）●deflate（zlib）●identity（不进行编码）
- 分割发送的分块传输编码
  - 在传输大容量数据时，通过把数据分割成多块，能够让浏览器逐步显示页面。这种把实体主体分块的功能称为分块传输编码（Chunked Transfer Coding）。 分块传输编码会将实体主体分成多个部分（块）。每一块都会用十六进制来标记块的大小，而实体主体的最后一块会使用“0(CR+LF)”来标记。
  - 使用分块传输编码的实体主体会由接收的客户端负责解码，恢复到编码前的实体主体。

http演变：1.0 1.1 2 3

1. HTTP/1.1、HTTP/2、HTTP/3 演变

说说 HTTP/1.1 相比 HTTP/1.0 提高了什么性能?

HTTP/1.1 相比 HTTP/1.0 性能上的改进：

使用 TCP 长连接的方式改善了 HTTP/1.0 短连接造成的性能开销。

支持 管道（pipeline）网络传输，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。

但 HTTP/1.1 还是有性能瓶颈：

请求 / 响应头部（Header）未经压缩就发送，首部信息越多延迟越大。只能压缩 Body 的部分；

发送冗长的首部。每次互相发送相同的首部造成的浪费较多；

服务器是按请求的顺序响应的，如果服务器响应慢，会招致客户端一直请求不到数据，也就是队头阻塞；

没有请求优先级控制；

请求只能从客户端开始，服务器只能被动响应。

那上面的 HTTP/1.1 的性能瓶颈，HTTP/2 做了什么优化?

HTTP/2 协议是基于 HTTPS 的，所以 HTTP/2 的安全性也是有保障的。

那 HTTP/2 相比 HTTP/1.1 性能上的改进：

- 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。
- 5. 服务器推送
    - HTTP/2 还在一定程度上改善了传统的「请求 - 应答」工作模式，服务不再是被动地响应，也可以主动向客户端发送消息。
    - 举例来说，在浏览器刚请求 HTML 的时候，就提前把可能会用到的 JS、CSS 文件等静态资源主动发给客户端，减少延时的等待，也就是服务器推送（Server Push，也叫 Cache Push）。

HTTP/2 有哪些缺陷？HTTP/3 做了哪些优化？

- 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。

HTTPS

http的不足

1. 通信使用明文（不加密），内容可能会被窃听
2. 不验证通信方的身份，客户端或者服务器都有可能遭遇伪装、即使是无意义的请求也全部接受
3. 无法证明报文的完整性，所以有可能已遭篡改（途中遭遇拦截并篡改内容的攻击成为中间人攻击）

加密处理

1. 通信加密
   - HTTP协议中没有加密机制，但可以通过和SSL（Secure Socket Layer，安全套接层）或TLS（Transport Layer Security，安全传输层协议）的组合使用，加密HTTP的通信内容。
   - 用SSL建立安全通信线路之后，就可以在这条线路上进行HTTP通信了。与SSL组合使用的HTTP被称为HTTPS（HTTP Secure，超文本传输安全协议）或HTTP over SSL。
2. 内容加密
   - 由于HTTP协议中没有加密机制，那么就对HTTP协议传输的内容本身加密。即把HTTP报文里所含的内容进行加密处理。
   - 在这种情况下，客户端需要对HTTP报文进行加密处理后再发送请求。

证书

虽然使用HTTP协议无法确定通信方，但如果使用SSL则可以。SSL不仅提供加密处理，而且还使用了一种被称为证书的手段，可用于确定方。

证书由值得信任的第三方机构颁发，用以证明服务器和客户端是实际存在的。另外，伪造证书从技术角度来说是异常困难的一件事。

证书认证

浏览器发起 HTTPS 请求时，服务器会返回网站的 SSL 证书，浏览器需要对证书做以下验证：

1、验证域名、有效期等信息是否正确。证书上都有包含这些信息，比较容易完成验证；

2、判断证书来源是否合法。每份签发证书都可以根据验证链查找到对应的根证书，操作系统、浏览器会在本地存储权威机构的根证书，利用本地根证书可以对对应机构签发证书完成来源验证；

3、判断证书是否被篡改。需要与 CA 服务器进行校验；

4、判断证书是否已吊销。通过CRL（Certificate Revocation List 证书注销列表）和 OCSP（Online Certificate Status Protocol 在线证书状态协议）实现，其中 OCSP 可用于第3步中以减少与 CA 服务器的交互，提高验证效率

HTTP+加密+认证+完整性保护 = HTTPS

- HTTPS并非是应用层的一种新协议。只是HTTP通信接口部分用SSL和TLS协议代替而已。
- 通常，HTTP直接和TCP通信。当使用SSL时，则演变成先和SSL通信，再由SSL和TCP通信了。
- 简言之，所谓HTTPS，其实就是身披SSL协议这层外壳的HTTP。
- SSL是独立于HTTP的协议，所以不光是HTTP协议，其他运行在应用层的SMTP和Telnet等协议均可配合SSL协议使用。可以说SSL是当今世界上应用最为广泛的网络安全技术。

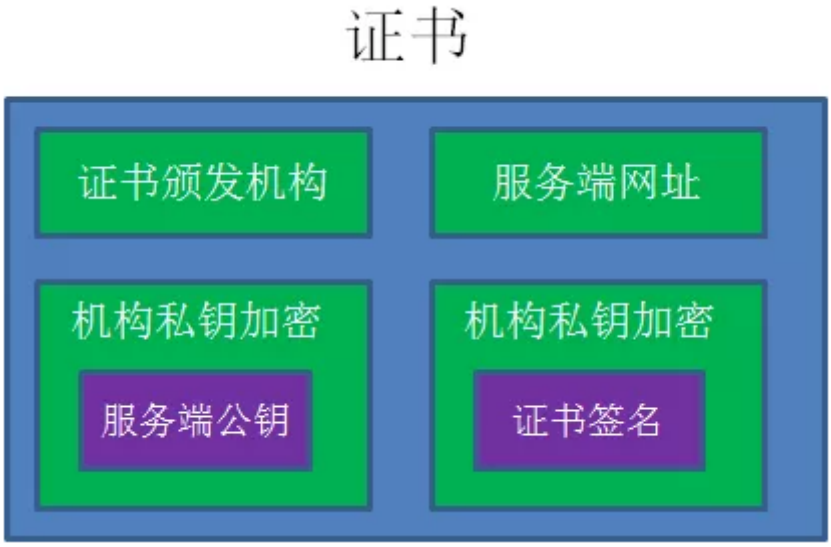
https流程

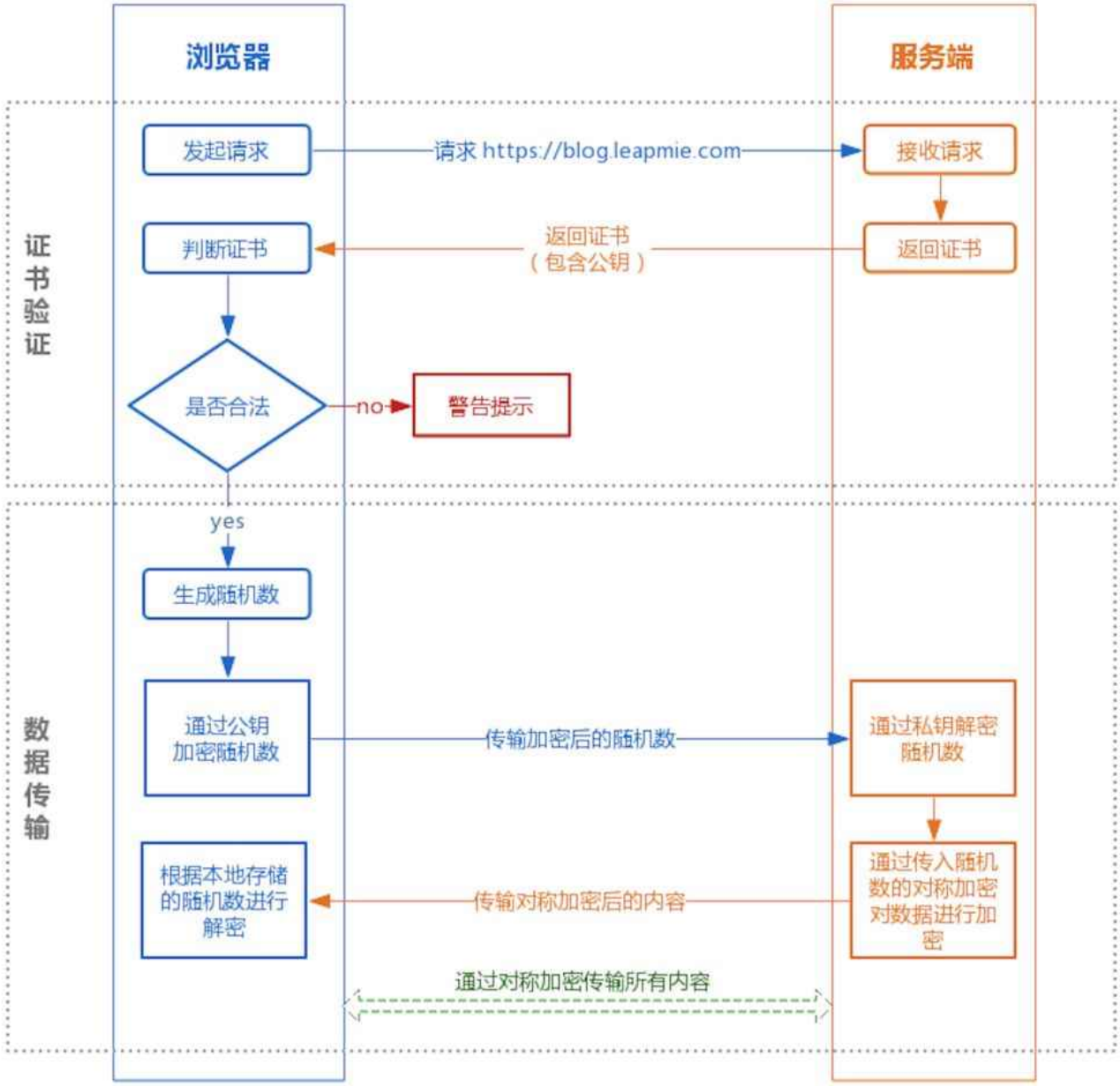
HTTPS 在内容传输的加密上使用的是对称加密，非对称加密只作用在证书验证阶段。

常用的非对称加密是RSA算法，对称加密是AES、RC4等，hash算法就是MD5

1. 服务器申请证书
   - 1.服务端首先把自己的公钥发给证书颁发机构，向证书颁发机构申请证书。
   - 2.证书颁发机构自己也有一对公钥私钥。机构利用自己的私钥来加密Key1，并且通过服务端网址等信息生成一个证书签名，证书签名同样经过机构的私钥加密。证书制作完成后，机构把证书发送给了服务端小红。
   - 3.当客户端请求通信的时候，服务端不再直接返回自己的公钥，而是把自己申请的证书返回
2. 证书验证阶段
   - 1）浏览器发起 HTTPS 请求
   - 2）服务端返回 HTTPS 证书（证书里有用证书私钥加密的网站地址、加密公钥、证书签名）
   - 3）客户端验证证书是否合法，如果不合法则提示告警（各大浏览器和操作系统已经维护了所有权威证书机构的名称和公钥。所以客户端只需要知道是哪个机构颁布的证书，就可以从本地找到对应的机构公钥，解密出证书签名。）
3. 数据传输阶段
   - 1）当证书验证合法后，客户端在本地生成随机数

- 2）通过服务端公钥加密随机数，并把加密后的随机数传输到服务端
- 3）服务端通过私钥对随机数进行解密
- 4）服务端通过客户端传入的随机数构造对称加密算法，对返回结果内容进行加密后传输





https问答

Q: HTTPS 为什么安全?

- : 因为 HTTPS 保证了传输安全，防止传输过程被监听、防止数据被窃取，可以确认网站的真实性。

Q: HTTPS 的传输过程是怎样的?

- A: 客户端发起 HTTPS 请求，服务端返回证书，客户端对证书进行验证，验证通过后本地生成用于改造对称加密算法的随机数，通过证书中的公钥对随机数进行加密传输到服务端，服务端接收后通过私钥解密得到随机数，之后的数据交互通过对称加密算法进行加解密。

Q: 为什么需要证书?

- A: 防止”中间人“攻击，同时可以为网站提供身份证明。

Q: 使用 HTTPS 会被抓包吗?

- A: 会被抓包，HTTPS 只防止用户在不知情的情况下通信被监听，如果用户主动授信，是可以构建“中间人”网络，代理软件可以对传输内容进行解密。

SSL的缺点

- SSL的慢分两种。一种是指通信慢。另一种是指由于大量消耗CPU及内存等资源，导致处理速度变慢。
- 和使用HTTP相比，网络负载可能会变慢2到100倍。除去和TCP连接、发送HTTP请求·响应以外，还必须进行SSL通信，因此整体上处理通信量不可避免会增加。
- 另一点是SSL必须进行加密处理。在服务器和客户端都需要进行加密和解密的运算处理。因此从结果上讲，比起HTTP会更多地消耗服务器和客户端的硬件资源，导致负载增强。

常见题目

加密算法

加密算法可以归结为三大类：哈希算法、对称加密算法、非对称加密算法。

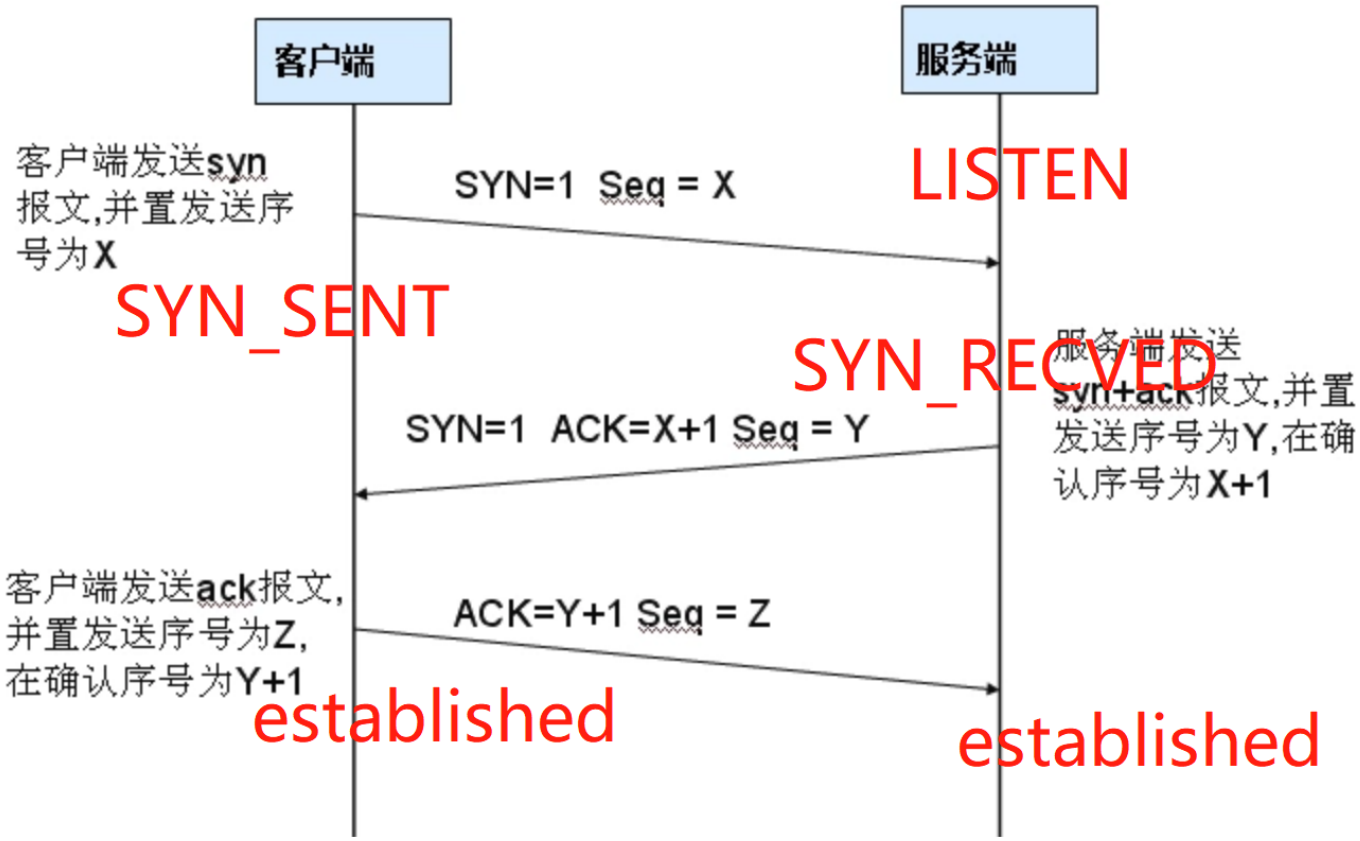
1. 哈希算法

- hash算法在信息安全领域起到了很重要的作用。
- 安全加密
  - 任意长度相同长度，不能反向推导
- 数据校验（信息摘要 签名）
  - 生成信息摘要，用以验证原信息的完整性和来源的可靠性。
- MD5算法：生成摘要
  - 1.收集相关业务参数，比如金额和目标账户。
  - 2.按照规则，**把参数名和参数值拼接成一个字符串，同时把给定的密钥也拼接起来**。之所以需要密钥，是因为攻击者也可能获知拼接规则。
  - 3.利用MD5算法，从原文生成哈希值。MD5生成的哈希值是128位的二进制数，也就是32位的十六进制数。
- MD5算法：验签
  - 1.发送方和请求方约定相同的字符串拼接规则，约定相同的密钥。
  - 2.第三方平台接到数据，按规则拼接业务参数和密钥，利用MD5算法生成Sign。
  - 3.用第三方平台自己生成的Sign和请求发送过来的Sign做对比，如果两个Sign值一模一样，则签名无误，如果两个Sign值不同，则信息做了篡改。这个过程叫做验签。

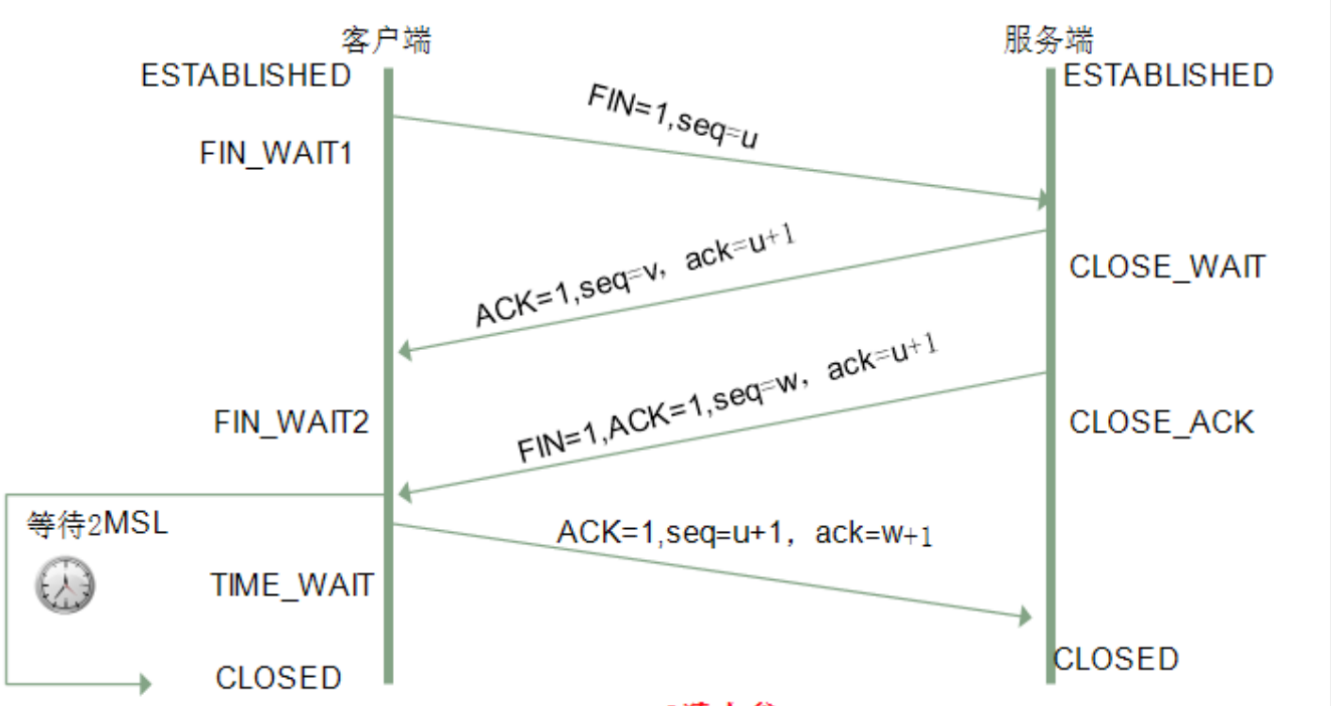
2. 对称加密算法
   - 在早期，人们使用DES算法进行加密解密；后来，人们觉得DES不够安全，发明了3DES算法；而如今，最为流行的对称加密算法是AES算法。
   - 对称算法的好处是加密解密的效率比较高。相应的，对称算法的缺点是不够安全，需要交换密钥
3. 非对称加密
   - 在加密解密的过程中，我们既可以使用公钥加密明文，使用私钥解密密文；也可以使用私钥加密明文，使用公钥解密密文。
   - RSA算法。

三次握手四次挥手

1. 信号
   - 序列号seq：占4个字节，用来标记数据段的顺序，TCP把连接中发送的所有数据字节都编上一个序号，第一个字节的编号由本地随机产生；给字节编上序号后，就给每一个报文段指派一个序号；序列号seq就是这个报文段中的第一个字节的数据编号。
   - 确认号ack：占4个字节，期待收到对方下一个报文段的第一个数据字节的序号；序列号表示报文段携带数据的第一个字节的编号；而确认号指的是期望接收到下一个字节的编号；因此当前报文段最后一个字节的编号+1即为确认号。
   - 确认ACK：占1位，仅当ACK=1时，确认号字段才有效。ACK=0时，确认号无效
   - 同步SYN：连接建立时用于同步序号。当SYN=1，ACK=0时表示：这是一个连接请求报文段。若同意连接，则在响应报文段中使得SYN=1，ACK=1。因此，SYN=1表示这是一个连接请求，或连接接受报文。SYN这个标志位只有在TCP建产连接时才会被置1，握手完成后SYN标志位被置0。
   - 终止FIN：用来释放一个连接。FIN=1表示：此报文段的发送方的数据已经发送完毕，并要求释放运输连接
2. 三次握手流程



3. 三次握手（Three-way Handshake）作用
   - 就是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备。
   - 实质上其实就是连接服务器指定端口，建立TCP连接，并同步连接双方的序列号和确认号，交换TCP窗口大小信息。
4. 四次挥手



为什么需要三次握手而不是两次

- 弄清这个问题，我们需要先弄明白三次握手的目的是什么，能不能只用两次握手来达到同样的目的。（需要确定客户端和服务单的发送接收能力都正常）
- 第一次握手：客户端发送网络包，服务端收到了。这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的。
- 第二次握手：服务端发包，客户端收到了。这样客户端就能得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。不过此时服务器并不能确认客户端的接收能力是否正常。
- 第三次握手：客户端发包，服务端收到了。这样服务端就能得出结论：客户端的接收、发送能力正常，服务器自己的发送、接收能力也正常。
- 因此，**需要三次握手才能确认双方的接收与发送能力是否正常**。
- 如果是用两次握手，则会出现下面这种情况：
- 如客户端发出连接请求，但因连接请求报文丢失而未收到确认，于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接，客户端共发出了两个连接请求报文段，其中第一个丢失，第二个到达了服务端，但是第一个丢失的报文段只是在某些网络结点长时间滞留了，延误到连接释放以后的某个时间才到达服务端，此时服务端误认为客户端又发出一次新的连接请求，于是就向客户端发出确认报文段，同意建立连接，
- **不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端忽略服务端发来的确认，也不发送数据，则服务端一致等待客户端发送数据，浪费资源。**

为什么需要四次挥手

- 建立一个连接需要三次握手，而终止一个连接要经过四次挥手（也有将四次挥手叫做四次握手的）。这由TCP的半关闭（half-close）造成的。所谓的半关闭，其实就是TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力。
- 当服务端收到客户端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。
- 但是关闭连接时，当服务端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉客户端，"你发的FIN报文我收到了"。只有等到我服务端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四次挥手。

客户端time_wait为什么要等待2MSL（最大报文段生存时间）

- 为了保证客户端发送的最后一个ACK报文段能够到达服务器。
- 因为这个ACK有可能丢失，从而导致处在LAST-ACK状态的服务器收不到对FIN-ACK的确认报文。服务器会超时重传这个FIN-ACK，接着客户端再重传一次确认，重新启动时间等待计时器。最后客户端和服务器都能正常的关闭。
- 假设客户端不等待2MSL，而是在发送完ACK之后直接释放关闭，一但这个ACK丢失的话，服务器就无法正常的进入关闭连接状态。

什么是半连接队列

- 服务器第一次收到客户端的 SYN 之后，就会处于 SYN_RCVD 状态，此时双方还没有完全建立其连接，服务器会把此种状态下请求连接放在一个队列里，我们把这种队列称之为半连接队列。
- 当然还有一个全连接队列，就是已经完成三次握手，建立起连接的就会放在全连接队列中。如果队列满了就有可能会出现丢包现象。
- 关于SYN-ACK 重传次数的问题：服务器发送完SYN-ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传。如果重传次数超过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。注意，每次重传等待的时间不一定相同，一般会是指数增长，例如间隔时间为 1s，2s，4s，8s.....

三次握手过程中可以携带数据吗?

- 其实第三次握手的时候，是可以携带数据的。但是，第一次、第二次握手不可以携带数据
- 假如第一次握手可以携带数据的话，如果有人要恶意攻击服务器，那他每次都在第一次握手中的 SYN 报文中放入大量的数据。因为攻击者根本就不理服务器的接收、发送能力是否正常，然后疯狂着重复发 SYN 报文的话，这会让服务器花费很多时间、内存空间来接收这些报文。也就是说，第一次握手不可以放数据，其中一个简单的原因就是会让服务器更加容易受到攻击了。
- 而对于第三次的话，此时客户端已经处于 ESTABLISHED 状态。对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据.

SYN攻击是什么?

- 服务器端的资源分配是在二次握手时分配的，而客户端的资源是在完成三次握手时分配的，所以服务器容易受到SYN洪泛攻击。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server则回复确认包，并等待Client确认，由于源地址不存在，因此Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络拥塞甚至系统瘫痪。SYN 攻击是一种典型的 DoS/DDoS 攻击。
- 检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击。在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。
  - netstat -n -p TCP | grep SYN_RECV
- 常见的防御 SYN 攻击的方法有如下几种：
  - 常见的防御 SYN 攻击的方法有如下几种：
  - 缩短超时（SYN Timeout）时间
  - 增加最大半连接数
  - 过滤网关防护
  - SYN cookies技术
  - 建立连接时SYN超时--server没有收到client发回来的ACK，则处在中间状态，一段时间之后会进行重发

浏览器一次请求过程

应用层发起请求

在传输层（tcp协议）把从应用层收到的数据（http请求报文）进行分割，并在各个报文上打上标记序号以及端口号后转发给网络层

在网络层（ip协议）增加作为通信目的的mac地址后转发给链路层

接收端的服务器在链路层接收到数据，按序往上层发送，一直到应用层。

> 发送端在层与层之间传输数据时，每经过一层时必定会被打上一个该层所属的首部信息。反之，接收端在层与层传输数据时，每经过一层时会把对应的首部消去

详细

1、客户端浏览器通过DNS解析到域名的IP地址220，通过这个IP地址找到客户端到服务器的路径。客户端浏览器通过三次握手发起一个HTTP会话，然后通过TCP进行封装数据包。

2、在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返

回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

3、客户端的网络层不用关系应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器

4、客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

web服务器

HTTP通信时，除客户端和服务器以外，还有一些用于通信数据转发的应用程序，例如代理、网关和隧道。它们可以配合服务器工作。

1. 代理
   - 代理是一种有转发功能的应用程序，它扮演了位于服务器和客户端“中间人”的角色，接收由客户端发送的请求并转发给服务器，同时也接收服务器返回的响应并转发给客户端。
   - 使用代理服务器的理由有：利用缓存技术（稍后讲解）减少网络带宽的流量，组织内部针对特定网站的访问控制，以获取访问日志为主要目的，等等。
   - 代理有多种使用方法，按两种基准分类。一种是是否使用缓存，另一种是是否会修改报文。
2. 网关
   - 网关是转发其他服务器通信数据的服务器，可以作为请求响应的唯一出口和入口。协议内容的转换，监控，分析都可以在网关中实现。路由也能算是网关，可以作为一个协议转换器。
3. 隧道
   - 隧道是在相隔甚远的客户端和服务器两者之间进行中转，并保持双方通信连接的应用程序。
   - 隧道可按要求建立起一条与其他服务器的通信线路，届时使用SSL等加密手段进行通信。
   - 隧道的目的是确保客户端能与服务器进行安全的通信。 隧道本身不会去解析HTTP请求。也就是说，请求保持原样中转给之后的服务器。隧道会在通信双方断开连接时结束。
4. 缓存
   - 缓存是指代理服务器或客户端本地磁盘内保存的资源副本。利用缓存可减少对源服务器的访问，因此也就节省了通信流量和通信时间。
   - 缓存服务器是代理服务器的一种，归类在缓存代理类型中。
5. 单台主机多个域名
   - 在相同的IP地址下，由于虚拟主机可以寄存多个不同主机名和域名的Web网站，因此在发送HTTP请求时，必须在Host首部内完整指定主机名或域名的URI。
   - 首部字段Host会告知服务器，请求的资源所处的互联网主机名和端口号。Host首部字段在HTTP/1.1规范内是唯一一个必须被包含在请求内的首部字段。

网络攻击

1. 中间人攻击
   - 拦截并篡改信息的手法
2. XSS：跨站脚本攻击
   - 跨站脚本攻击（Cross-Site Scripting, XSS），可以将代码注入到用户浏览的网页上，这种代码包括 HTML 和 JavaScript。
   - 网页中嵌入恶意脚本，当用户使用浏览器浏览这些被嵌入恶意脚本的网页的时候，脚本就会在我们的浏览器中执行。xss攻击的核心方式是 脚本。
   - 解决：
     - 过滤特殊字符
3. CSRF：跨站请求伪造
   - 攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己曾经认证过的网站并执行一些操作（如发邮件，发消息，甚至财产操作如转账和购买商品）。由于浏览器曾经认证过，所以被访问的网站会认为是真正的用户操作而去执行。
   - XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户浏览器的信任。
   - 解决：
     - 1. 检查 Referer 首部字段
     - 2. 添加校验 Token
     - 3.输入验证码
4. SQL 注入攻击
   - 服务器上的数据库运行非法的 SQL 语句，主要通过拼接来完成。
   - 防范手段
     - 1. 使用参数化查询
     - 2. 单引号转换
5. 拒绝服务攻击（denial-of-service attack，DoS）
   - 亦称洪水攻击，其目的在于使目标电脑的网络或系统资源耗尽，使服务暂时中断或停止，导致其正常用户无法访问。
6. 分布式拒绝服务攻击（distributed denial-of-service attack，DDoS）
   - 指攻击者使用两个或以上被攻陷的电脑作为“僵尸”向特定的目标发动“拒绝服务”式攻击。
   - 对目标网站在较短的时间内发起大量请求，大规模消耗目标网站的主机资源，让它无法正常服务。