✨ OSI七层协议、TCP/IP四层协议

- **OSI七层协议：**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四层协议：**网络接口层、网络层、传输层、应用层
- **五层协议：**物理层、数据链路层、网络层、传输层、应用层

网络七层模型是一个标准，网络四层模型是一个实现的应用模型。

分层的目的：可以将庞大而复杂的问题转化为若干较小的局部问题，而这些较小为局部问题就比较易于研究和处理。

各层的作用：

- **物理层**：考虑怎样能够在连接各种计算机的传输媒体上传输数据比特流，尽可能的屏蔽掉这些传输媒体和通信手段的差异。
- **数据链路层：**在两个相邻节点的链路上传输数据时，数据链路层将网络层交下来的IP数据报封装成帧，在两个相邻节点间的链路上传送帧。
- **网络层：**负责为分组交换网上不同主机提供通信服务。在发送数据时，网络层将传输层产生的报文段或数据报封装成分组或包进行传送。网络层的另一个任务就是选择合适的路由，使源主机传输层所传下来的分组能够通过网络中的路由器找到目的主机。
- **传输层：**负责向两台主机中进程之间的通信提供通用的数据传输服务。应用进程利用该服务传输应用层报文。所谓通用的，是指并不针对某个特定的网络应用，而是多种应用可以使用同一个运输层服务。由于一台主机可同时运行多个进程，因此传输层具有复用和分用的功能，复用就是多个应用进程可同时使用下面传输层的服务，分用和复用相反，是传输层把收到的信息分别交付上面应用层中的相应进程。
- **应用层：**通过应用进程间的交互来完成特定网络应用。应用层协议定义的是应用进程间通信和交互的规则。对于不同的网络应用需要有不同的应用层协议。

✨ 计算机网络性能指标

- **比特率：**单位时间内传输的比特数（单位：bps，b/s）
- **带宽：**单位时间内通信链路能够通过的数据量（b/s）
- **吞吐量：**单位时间内通过某个网络的实际数据量（b/s）
- **时延：**数据从网络一端传送到另一端所需的时间（总时延=发送时延+传播时延+处理时延+排队时延）
- **往返时间RTT：**从发送方发送数据开始，到发送方收到来自接收方的确认总共经历的时间

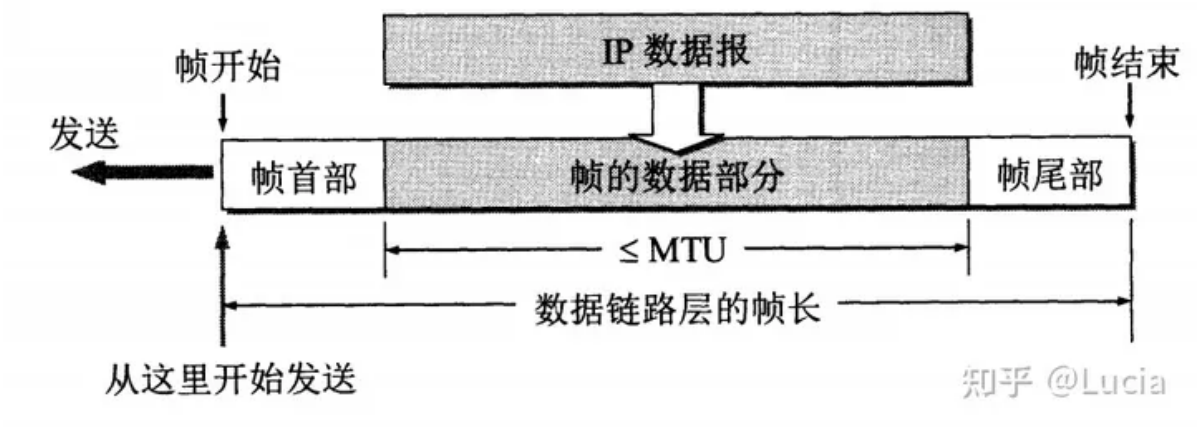
比特率和波特率：

- 数据传输速率（比特率）：单位时间内传输的二进制位数
- 信号传输速率（波特率）：单位时间内通过信道传输的码元个数

比特率=波特率×log2N比特率=波特率×log2﷟（N﷟为一个码元携带的离散化电平个数）

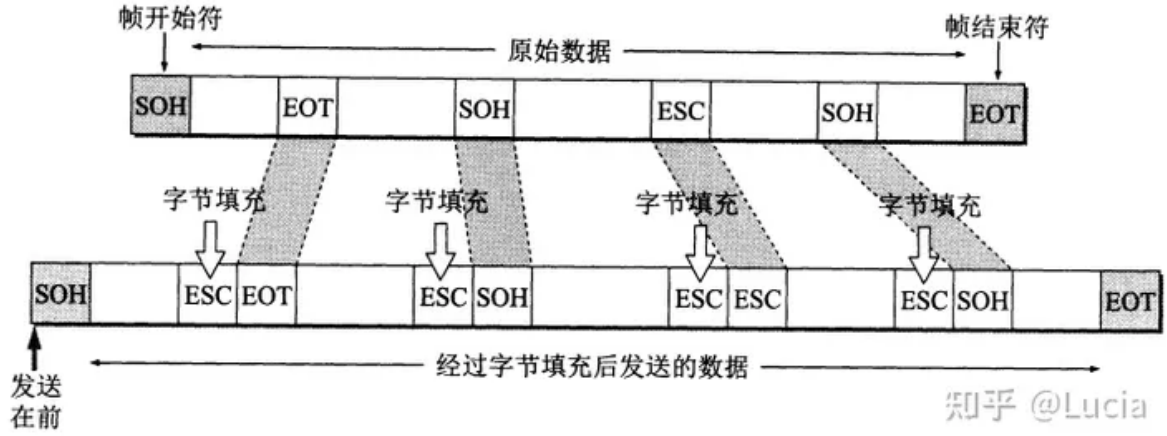
✨ 数据链路层的三个基本功能

1. **封装成帧：**将IP数据报前后分别添加首部和尾部构成帧（SOH、EOT）。首部和尾部的一个重要作用：帧定界，接收端在收到物理层上交的比特流后，能够根据首部和尾部的标记，从收到的比特流中识别帧的开始和结束。



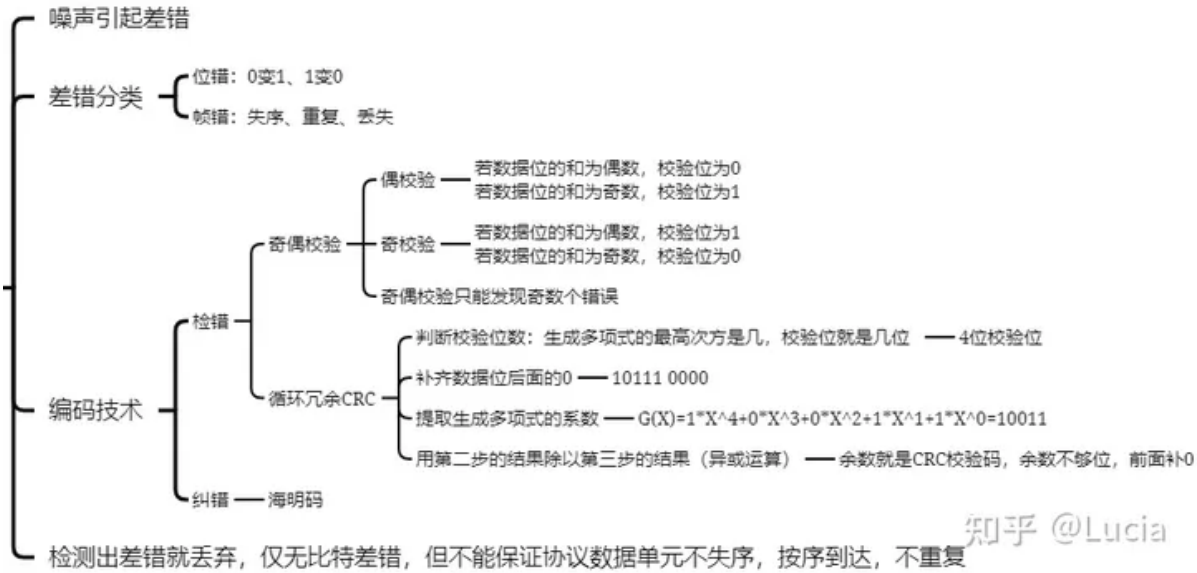
MTU：最大传输单元，数据部分长度上限

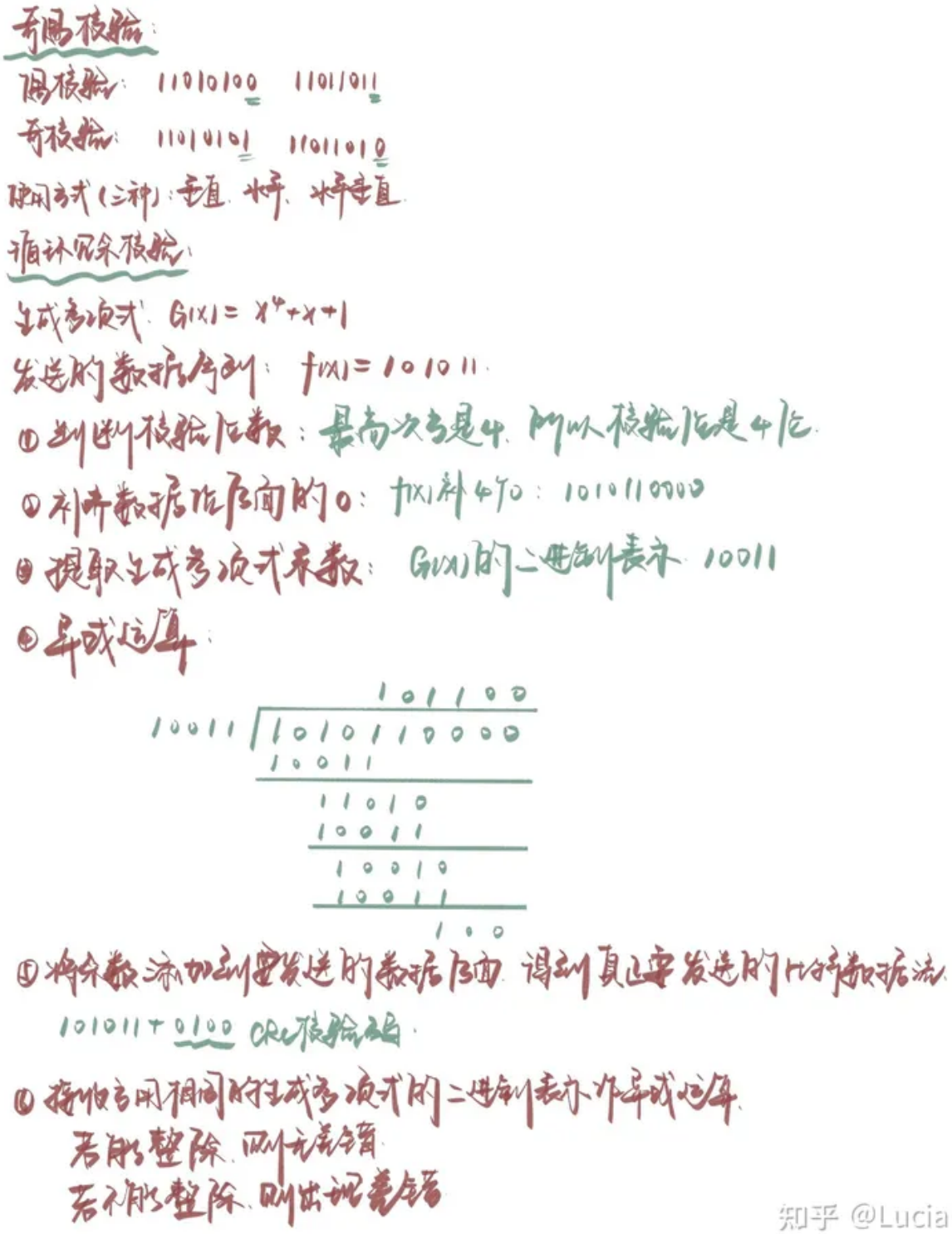
2. **透明传输：**发送端的数据链路层在数据中出现控制字符"SOH"和"EOT"的前面插入一个转义字符ESC。



透明：无论什么样的比特组合的数据，都能够按照原样没有差错地通过数据链路层

3. **差错控制：**



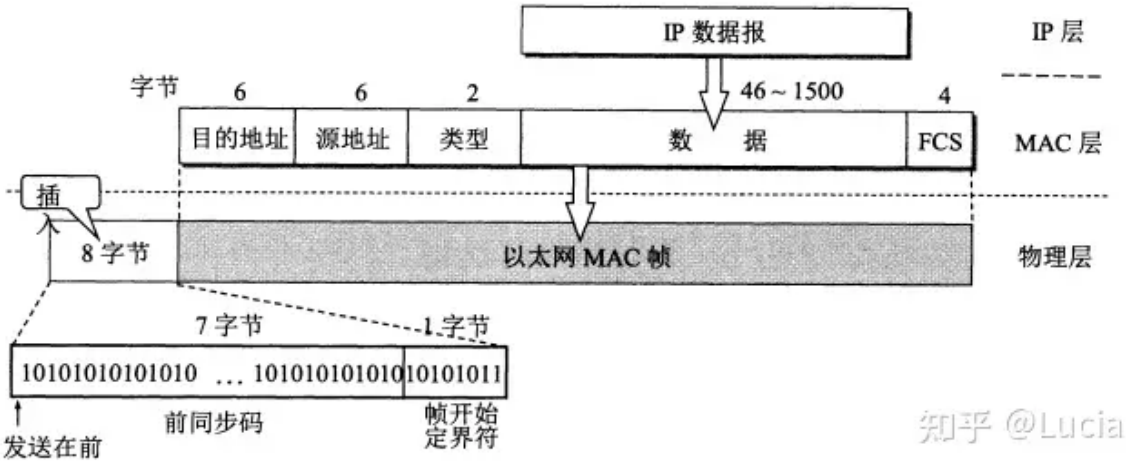


虽然数据链路层可以通过多种方式来检测和纠正传输过程中可能出现的错误，但它并不是完全无差错的。

✨ 数据链路层的实现方式

- 点对点网络（P2P）：使用一对一的点对点通信方式，没有信道竞争，会浪费一些带宽。
- 广播网络：使用一对多的广播通信方式，通过向所有站点发送分组的方式传输信息。

✨MAC帧格式



- 类型字段用来标志上一层使用的是什么协议，以便把收到的MAC帧的数据上交给上一层的这个协议。
- 前同步码用来迅速实现MAC帧的比特同步，帧开始定界符表示后面的信息就是MAC帧。

MAC地址（48位）

IEEE 的注册管理机构 RA(Registration Authority)负责向厂家分配地址字段的前三个字节(即高位24位)，而后三个字节(即低位24位)由厂家自行指派，称为扩展标识符，必须保证生产出的适配器没有重复地址。一个地址块可以生成 224224 个不同的地址。

- **个人/组 (I/G, Individual/Group)位：**第一个字节的最低位，用来确定地址是单播还是多播地址。当 I/G位=0 时，地址字段表示一个单站地址。当 I/G位=1时，表示组地址，用来进行多播（以前曾译为组播）。此时， IEEE 只分配地址字段前三个字节中的23位。
  当I/G位分别为0和1时，一个地址块可分别生成224224个单个站地址和224224个组地址。
  所有48位都为1时，为广播地址。只能作为目的地址使用 。
- **全局/本地 (G/L, Global/Local)位：**第一个字节的第七位，用于确定该地址是全局管理的还是本地管理的。当G/L位=0时，是全球管理（保证在全球没有相同的地址），厂商向IEEE购买的OUI都属于全球管理。当G/L位=1时，是本地管理， 这时用户可任意分配网络上的地址。

✨ 以太网广播方式中的碰撞问题：CSMA/CD协议

- **多点接入**：表示许多计算机以多点接入的方式连接在一根总线上
- **载波监听**：每一个站在发送数据之前要先检测一下总线上是否有其他计算机在发送数据，如果有，则暂时不要发送数据，以免发生碰撞
- **碰撞检测**：每一个正在发送的站，一旦发现总线上出现了碰撞，就要立即停止发送，免得继续浪费网络资源，然后等待一段随机事件后再次发送

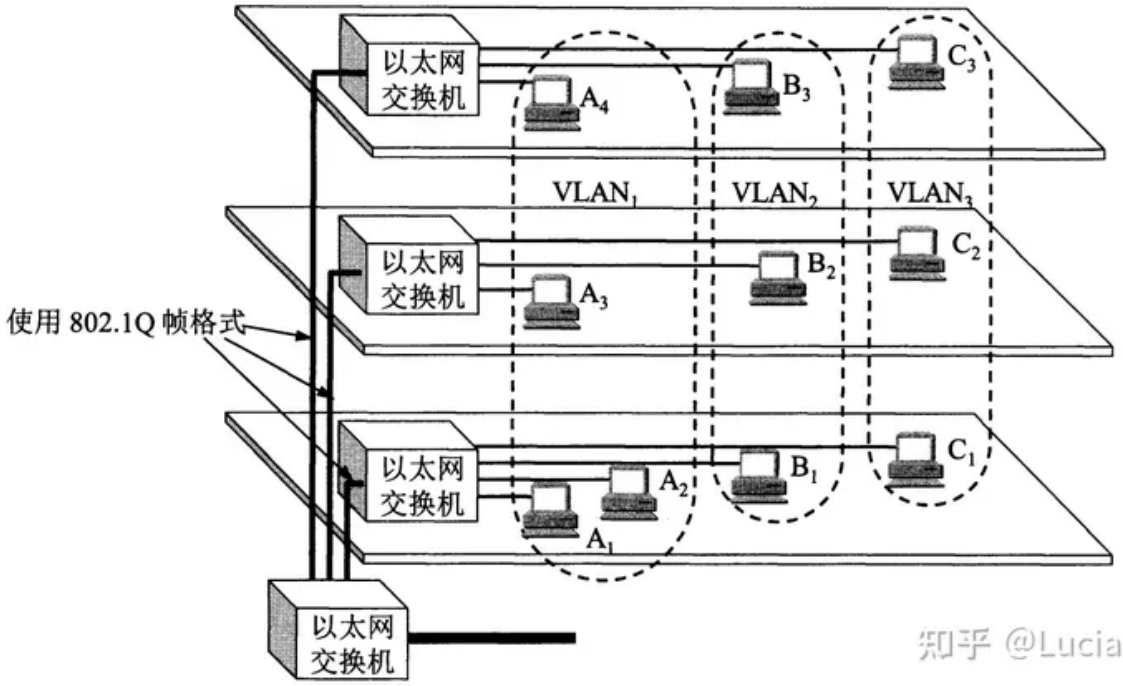
以太网使用**截断二进制指数退避算法**让发生碰撞的站在停止发送数据后，不是等待信道变为空闲后就立即再发送数据，而是推迟（退避）一个随机时间。

✨ 虚拟局域网

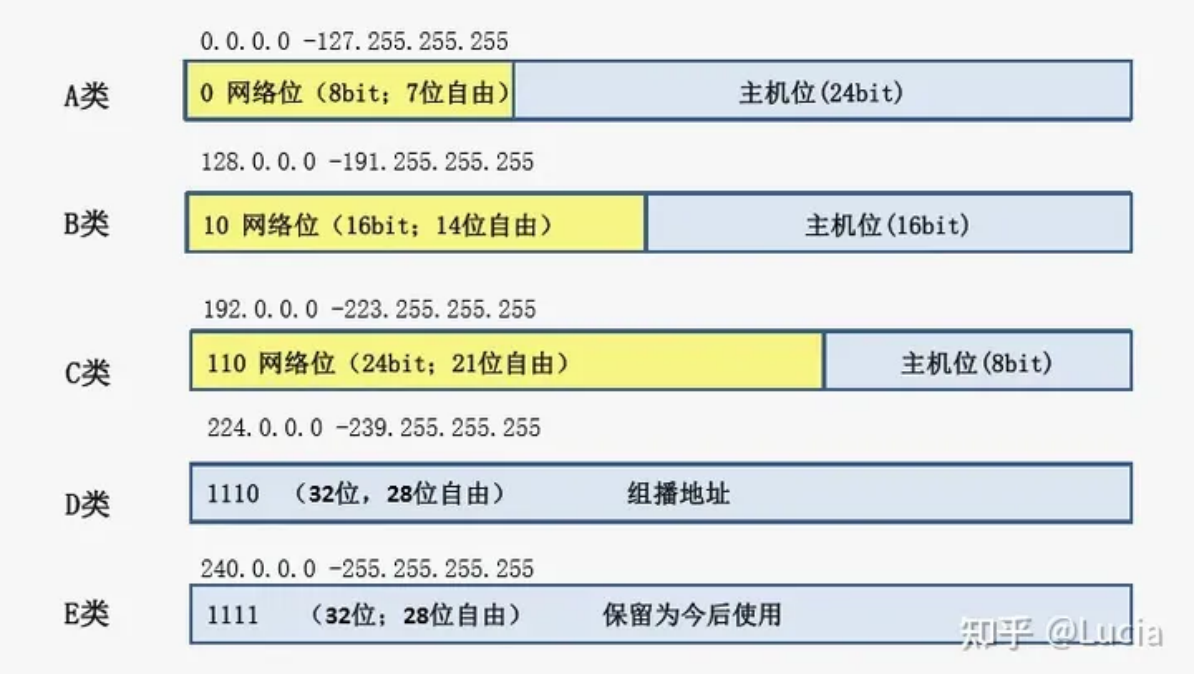
虚拟局域网VLAN是由一些局域网网段构成的与物理位置无关的逻辑组，而这些网段具有某些共同的需求。虚拟局域网协议允许在以太网帧格式中插入一个4字节的标识符，称为**VLAN标记**，用来指明发送该帧的计算机属于哪一个虚拟局域网。插入VLAN标记得出的帧称为**802.1Q帧**。

VLAN定义方法

- 基于端口的VLAN
- 基于MAC地址的VLAN
- 基于IP地址的VLAN
- 基于策略的VLAN



✨ IP地址分类



三种IP地址：`{<网络号>,<主机号>}`、`{<网络号>,<子网号>,<主机号>}`、`{<网络前缀>,<主机号>}`

一般不使用的特殊的IP地址：

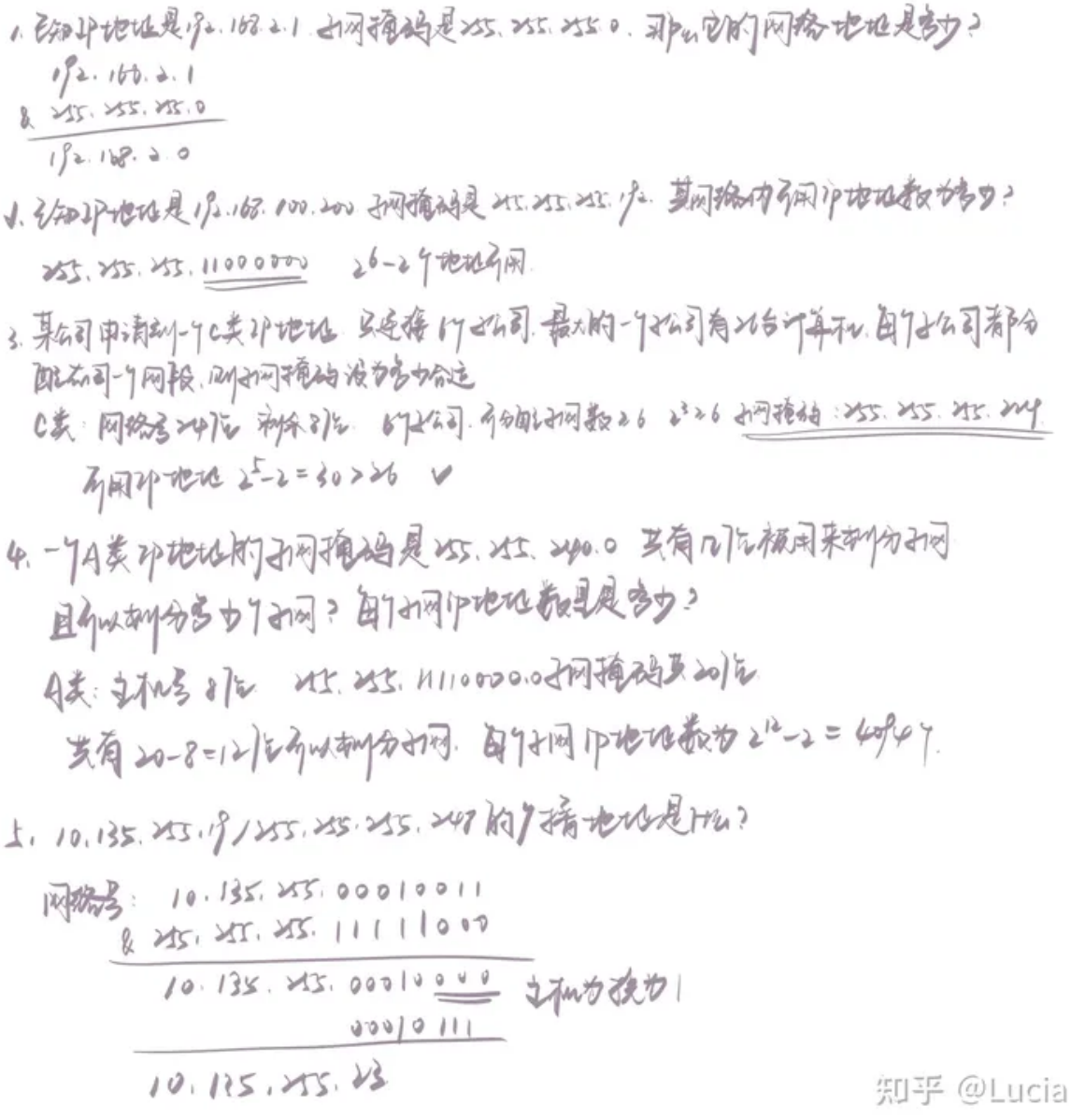
网络号	主机号	源地址使用	目的地址使用	代表的意思
0	0	可以	不可以	本网络上的本主机（DHCP协议使用）

网络号	主机号	源地址使用	目的地址使用	代表的意思
0	host-id	可以	不可以	在本网络上的某台主机host-id
全1	全1	不可以	可以	只在本网络上进行广播（各路由器均不转发）
net-id	全1	不可以	可以	对net-id上的所有主机进行广播
127	非全0非全1	可以	可以	用作本地软件环回测试使用

专用地址：只能用作本地地址，不能作为全球地址

- `10.0.0.0 – 10.255.255.255`
- `172.16.0.0 – 172.31.255.255`
- `192.168.0.0 – 192.168.255.255`

相关计算题



- 主机号全0：在一个网络段中主机号全为0代表这个网络段本身，称之为：网络号。这个地址是不可以分配给主机的。
- 主机号全1：主机号全1的地址是广播地址，给这个地址发数据包，这个网络内的所有主机都能收到。

✨ IP协议配套的三个协议

1. **地址解析协议(ARP)**：从网络层使用的IP地址，解析出在数据链路层使用的硬件地址；
2. **网际控制报文协议(ICMP)**：允许主机或路由器报告差错情况和有关异常情况的报告（应用：Ping用来测试两个主机之间的连通性，tracert用来跟踪一个分组从源点到终点的路径）；
3. **网际组管理协议(IGMP)**：通知路由器特定组播的信息。

✨ ARP协议的工作原理

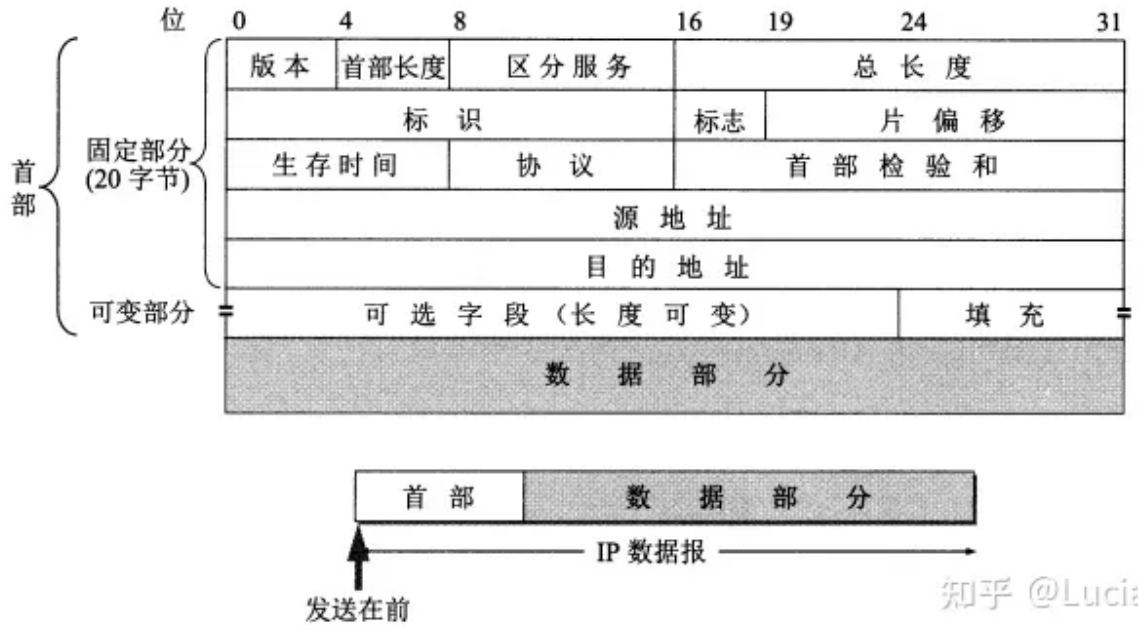
每一个主机都设有一个ARP高速缓存，里面有所在的局域网上的各主机和路由器的IP地址到硬件地址的映射表：`<IP address, MAC address, TTL>`（TTL：地址映射有效时间）

当主机A欲向本局域网上的某个主机B发送IP数据报时，就先在其ARP高速缓存中查看有无主机B的IP地址：

- 如果有，就可查出其对应的硬件地址，再将此硬件地址写入MAC帧，然后通过局域网将该MAC帧发往此硬件地址；
- 如果没有，ARP进程在本局域网上广播发送一个ARP请求分组，收到ARP响应分组后，将得到的IP地址到硬件地址的映射写入ARP高速缓存

为了减少网络上的通信量，主机A在发送其ARP请求分组时，就将自己的IP地址到硬件地址的映射写入ARP请求分组。当主机B收到A的ARP请求分组时，就将主机A的这一地址映射写入主机B自己的ARP高速缓存中，这对主机B以后向A发送数据报时就更方便了。

✨ IP数据报



- 一个 IP 数据报（分组）由首部和数据两部分组成。首部的前一部分是固定长度，共 20 字节，是所有 IP 数据报必须具有的。在首部的固定部分的后面是一些可选字段，其长度是可变的。
- 首部长度——占4位，可表示的最大数值是15个单位（一个单位是4字节）因此IP的首部长度最大值是60字节。

- 总长度——占16位，指首部和数据之和的长度，单位为字节，因此数据报的最大长度为65535字节。总长度必须不超过最大传输单元MTU。
- 标志——占3位，最低位`MF=1`表示后面还有分片，`MF=0`表示最后一个分片。中间位`DF=0`时才允许分片。
- 片偏移——占13位，较长的分组在分片后某片在原分组中的相对位置。
- 生存时间——占8位（TTL），数据报在网络中可通过的路由器数的最大值。
- 首部校验和——占16位，只检验数据报的首部不检验数据部分。采用16位二进制反码求和算法。出错则丢弃该数据报。

✨ 路由表

在路由表中，每一条路由：`(目的网络地址，下一跳地址)`。根据目的网络地址就能确定下一跳路由器，这样做的结果是IP数据报最终一定可以知道目的主机所在的目的网络上的路由器（可能要通过多次的间接交付）；只有到达最后一个路由器时，才试图向目的主机进行直接交付。路由表没有给分组指明到某个网络的完整路径。 在到达下一跳路由器后，再继续查找其路由表，知道再下一步应当到哪一个路由器。这样一步一步地查找下去，直到最后到达目的网络。

两类特殊路由：

- 特定主机路由：为特定的目的主机指明一个路由。
- 默认路由：路由器还可采用默认路由以减少路由表所占用的空间和搜索路由表所用的时间。这种转发方式在一个网络只有很少的对外连接时是很有用的。

路由器分组转发算法

1. 从数据报首部提取目的主机的IP地址D，得出目的网络地址为N。
2. 若网络N与此路由器直接相连，则把数据报直接交付目的主机D。否则是间接交付，执行3。
3. 若路由表中有目的地址为D的特定主机路由，则把数据报传送给路由表指明的下一跳路由器，否则执行4。
4. 若路由表中有到达网络N的路由，则把数据报传送给路由表指明的下一跳路由器，否则执行5。
5. 若路由表中有一个默认路由，则包数据报传送给路由表中所指明的默认路由器，否则执行6。
6. 报告转发分组出错。

注意：IP数据包转发过程中，IP报文首部中的源和目的IP地址保持不变，但每条链路上数据帧的源和目的MAC地址是不同的。

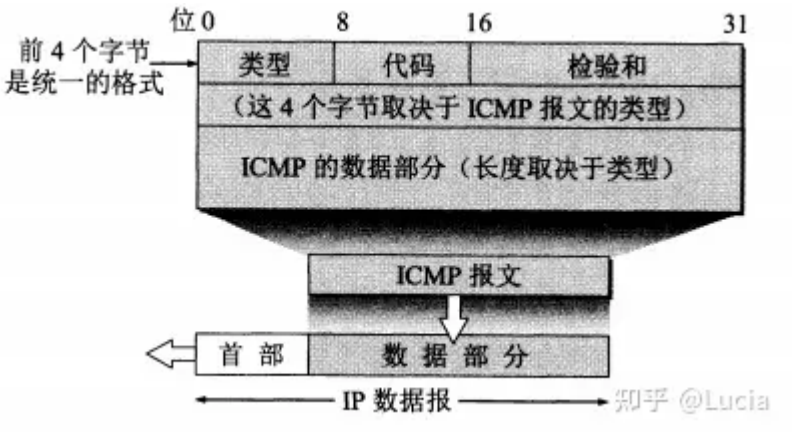
✨ ICMP报文

ICMP 报文作为 IP 层数据报的数据，加上数据报的首部，组成 IP 数据报发送出去。

种类：

- ICMP差错报告报文：终点不可达、源点抑制、时间超过、参数问题、改变路由（重定向）。
- ICMP询问报文：回送请求和回答报文、时间戳请求和回答报文。

报文格式：



✨ 路由选择协议

Internet采用**分层次**的路由选择协议。

自治系统AS有权自主地决定在本系统内采用何种路由协议。一个AS所有路由器在本AS内都必须是连通的。

自治系统之间的路由选择叫做域间路由选择（BGP-4）；自治系统内部的路由选择叫做域内路由原则（RIP、OSPF）。

基于距离向量的RIP协议

- RIP协议要求网络中的每一个路由器都要维护从它自己到其他每一个目的网络的距离记录。
- 仅和相邻路由器交换信息，交换的信息是本路由器当前所知道的全部信息。定时交换路由信息。RIP协议使用传输层的UDP进行传输。
- 特点：好消息传播得快，坏消息传播得慢。当网络出现故障时，要经过比较长的时间才能将此信息传送到所有的路由器。限制网络规模（最大距离为15）

距离向量算法

收到相邻路由器(其地址为X)的一个RIP报文：

1. 先修改此RIP报文中的所有项目：把下一跳字段中的地址都改为X，并把所有的距离字段的值加一。
2. 对修改后的RIP报文中的每一个项目重复以下步骤：
   若项目中的目的网络N不在路由表中，则把该项目加到路由表中，否则：

- 若下一跳字段给出的路由器地址是同样的X，则把收到的项目替换原路由表中的项目；
- 否则（即这个项目是到目的网络N，但是下一跳不是X）若收到项目中的距离小于路由表中的距离，则进行更新，否则，什么也不做。
- 若三分钟还没有收到相邻路由器的更新路由表，则把此相邻路由器记为不可达路由器，即将距离设置为16。
- 返回。

基于链路状态的OSPF协议

交换的信息是与本路由器相邻的所有路由器的链路状态，但只是路由器所知道的部分路由信息。使用洪泛法向本AS内所有路由器发送信息。只有当链路状态发生变化时，才发送信息。采用IP协议作为通信协议。

✨ 内部网关协议RIP和OSPF的区别

(路由信息协议RIP，开放最短路径优先协议OSPF)

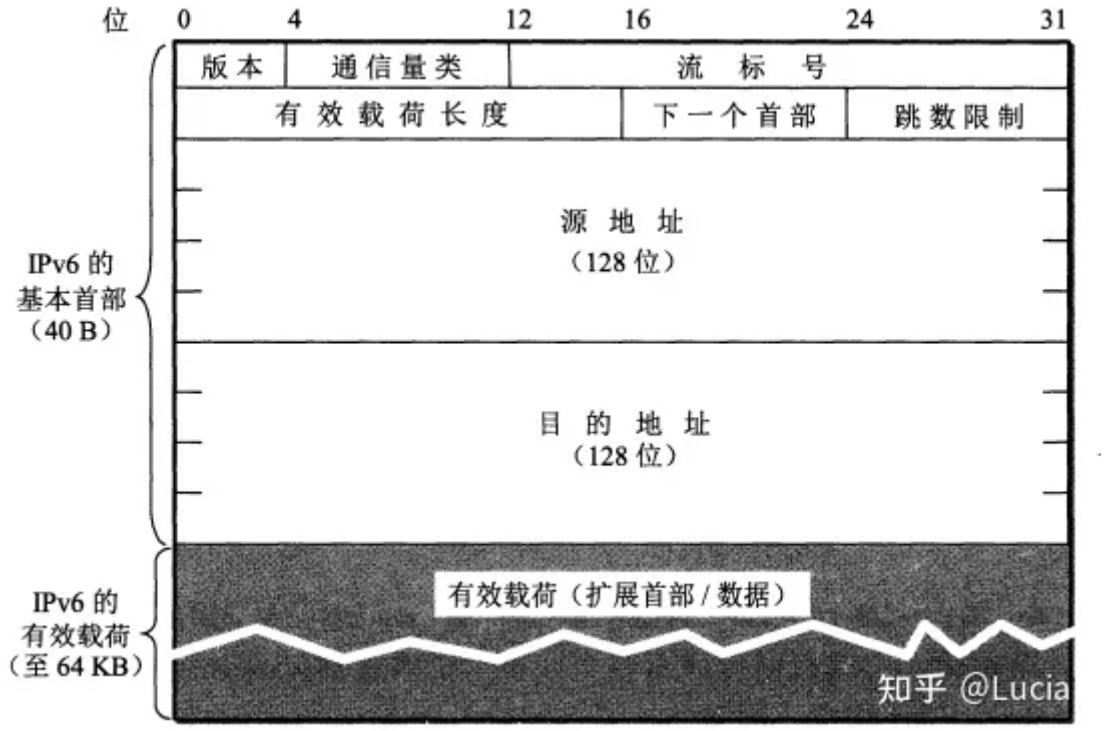
1. RIP仅和相邻路由器交换信息，OSPF采用洪泛法向本自治系统AS中所有路由器发送信息；
2. RIP交换的信息是本路由器当前所知道的全部信息，OSPF发送的信息是与本路由器相邻路由器的链路状态（当前已知的部分信息）OSPF所有路由器最终会建立一个包含全网拓扑结构图信息的链路状态库，而RIP不可以；
3. RIP定时交换路由信息，OSPF只有当链路状态发生变化时才会向所有路由器发送信息；
4. RIP使用UDP进行传送，OSPF直接使用IP数据报传送；
5. RIP限制网络规模，最大距离15；

✨ 网络地址转换NAT

需要在专用网连接到因特网的路由器上安装NAT软件，装有NAT软件的路由器叫做NAT路由器，它至少有一个有效的外部全球地址。所有使用本地地址的主机在和外界通信时都要在NAT路由器上将其本地地址转换成全球地址才能和因特网连接。

通过 NAT 路由器的通信必须由专用网内的主机发起。 专用网内部的主机不能充当服务器用， 因为互联网上的客户无法请求专用网内的服务器提供服务。 （NAPT加端口号）

✨ IPv6



IPv6数据报由两大部分组成：基本首部和有效载荷。有效载荷允许有零个或多个扩展首部，再后面是数据部分。

IPv6将首部长度变为固定的40字节，称为基本首部。将首部中不必要的功能取消了，使得IPv6首部的字段数减少到了8个。

- 取消了首部长度字段，因为首部长度是固定的40字节。
- 取消了服务类型字段。
- 取消了总长度字段，改用有效载荷长度字段。

- 取消了标识、标志和片偏移字段，含在分片扩展首部中。
- TTL字段改称为跳数限制字段。
- 取消了协议字段，改用下一个首部字段。
- 取消了检验和字段。
- 取消了选项字段，而用扩展首部来实现选项功能。

IPv6中，每个地址占128位，使用冒号十六进制记法。

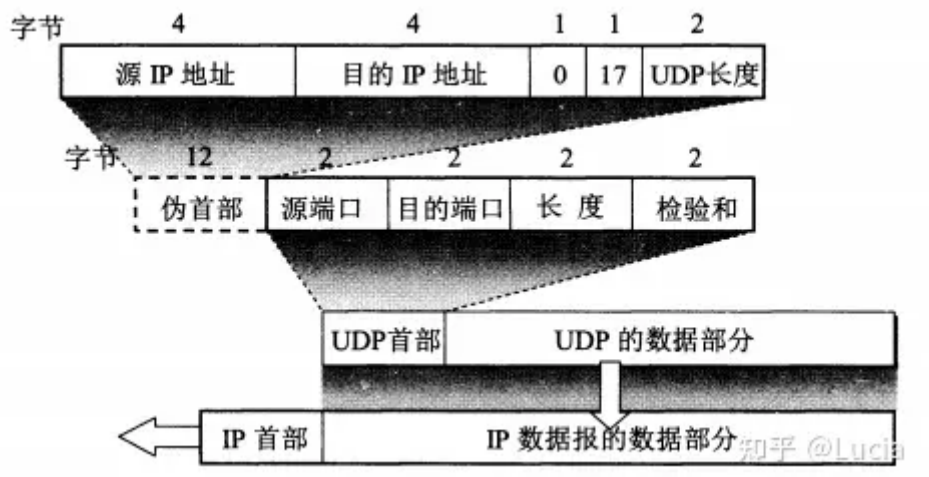
IPv4向IPv6过渡：

- 双协议栈：部分主机装有两个协议栈，一个IPv4和一个IPv6，同时具有两种IP地址，一个IPv6地址和一个IPv4地址。
- 隧道技术：在IPv6数据报要进入IPv4网络时，把IPv6数据报封装成为IPv4数据报，整个IPv6数据报变为IPv4数据报的数据部分。

✨ 常用熟知端口号

应用程序	FTP	TELNET	SMTP	DNS	TFTP	HTTP	SNMP	HTTPS
端口号	21	23	25	53	69	80	162	443

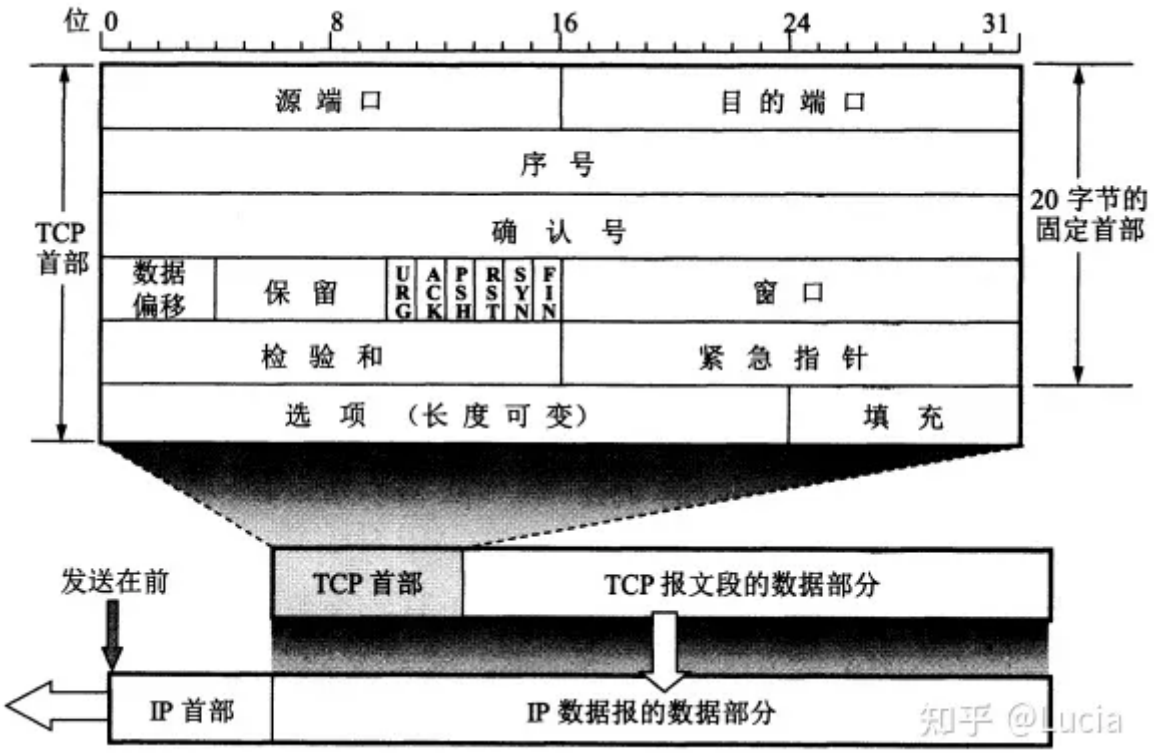
✨ UDP



用户数据报UDP有两个字段：数据字段和首部字段。首部字段有8个字节，由4个字段组成，每个字段都是两个字节。

在计算检验和时，临时把“伪首部”和UDP用户数据报连接在一起，伪首部仅仅是为了计算检验和。

✨ TCP



- 序号（32bit）：传输方向上字节流的字节编号。初始时序号会被设置一个随机的初始值（ISN），之后每次发送数据时，序号值=ISN + 数据在整个字节流中的偏移。用于解决网络包乱序问题。
- 确认号（32bit）：接收方对发送方TCP报文段的响应，其值是收到的序号值 + 1。
- 首部长（4bit）：标识首部有多少个4字节 * 首部长，最大为15，即60字节。
- 标志位（6bit）：
  - URG：标志紧急指针是否有效；
  - ACK：标志确认号是否有效（确认报文段），用于解决丢包问题；
  - PSH：提示接收端立即从缓冲读走数据；
  - RST：标识要求对方重新建立连接（复位报文段）；
  - SYN：表示请求建立一个连接（连接报文段）；
  - FIN：表示关闭连接（断开报文段）；
- 窗口（16bit）：接收窗口。用于告知对方（发送方）本方的缓冲还能接收多少字节数据。用于流量控制。
- 校验和（16bit）：接收端用CRC检验整个报文段有无损坏。

✨ TCP和UDP的区别

- UDP提供无连接、尽最大可能的交付，没有流量控制、拥塞控制，支持一对一、一对多、多对一的交互通信，速度快， 是面向报文的协议，一次发送或交付一个完整报文。
- TCP面向连接，提供可靠交付，有流量控制、拥塞控制，每一条TCP连接只能是一对一的，不提供多播和组播服务。速度慢，是面向流的协议。消息在传输过程中可能会乱序，后发送的消息可能会先到达，TCP会对其进其重排序，UDP不会。

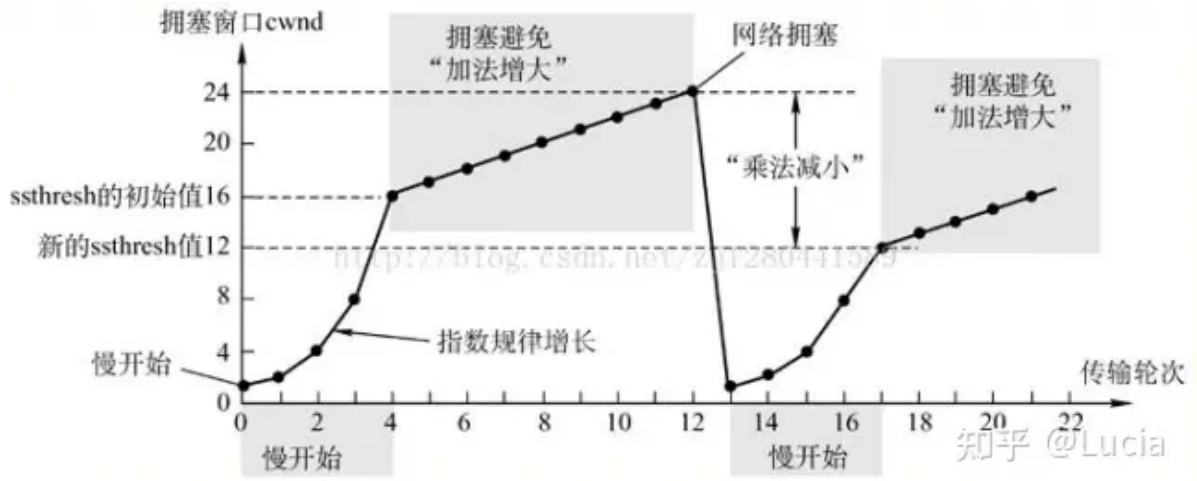
✨ TCP的流量控制

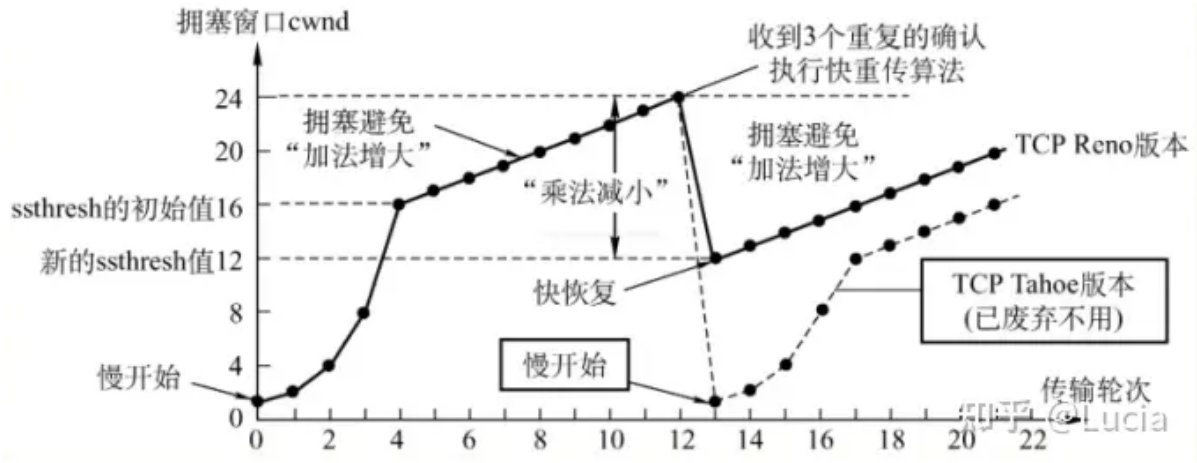
- 目的是接收方通过TCP头窗口字段告知发送方本方可接收的最大数据量，用以解决发送速率过快的导致接收方不能接收的问题。所以流量控制是点对点控制。
- TCP是双工协议，双方可以同时通信，所以发送方接收各自维护一个发送窗和接收窗。
  - 发送窗：用来限制发送方可以发送的数据大小，其中发送窗口的大小由接收端返回的TCP报文段中窗口字段来控制，接收方通过此字段告知发送方自己的缓冲大小。
  - 接收窗：用来标记可以接收的数据大小。
- TCP是流数据，发送出去的数据流可以被分为以下四部分：已发送且被确认部分 | 已发送未被确认部分 | 未发送但可发送部分 | 不可发送部分，其中发送窗 = 已发送未被确认部分 + 未发送但可发送部分，接收到的数据流可分为：已接收部分 | 未接收但准备接收部分 | 未接收不准备接收部分，接收窗 = 未接收但准备接收部分。
- 发送窗内数据只有当接收到接收端某段发送数据的ACK响应时才移动发送窗，左边缘紧贴刚被确认的数据。接收窗也只有接收到数据且最左侧连续是才移动接收窗口。

✨ TCP的拥塞控制

- **cwnd**：拥塞窗口
- **ssthresh**：慢开始门限（cwnd<ssthresh时使用慢开始算法，cwnd>ssthresh时使用拥塞避免算法）

1. **慢开始**：TCP开始发送报文段时先设置cwnd=1，使得发送方在开始时只发送一个报文段，之后每收到一个报文段的确认，拥塞窗口cwnd大小加1，每经过一个传输轮次（一个传输轮次所经历的时间是往返时间RTT），拥塞窗口cwnd加倍；
2. **拥塞避免**：当拥塞窗口cwnd增长到慢开始门限值ssthresh时，执行拥塞避免算法，把拥塞窗口控制为按线性规律增长。当网络中出现超时，发送方判断为网络拥塞，调整门限值为ssthresh = cwnd / 2，同时设置cwnd = 1，进入慢开始阶段；
3. **快重传**：当发送方一连收到3个对同一个报文段的重复确认时，为了避免让发送方产生超时误认为网络发生拥塞（报文段丢失，并未发生拥塞），错误的启动慢开始算法，降低传输效率，启用快重传方法让发送方尽早知道发生了个别报文段的丢失。此时发送方立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。
4. **快恢复**：发送方知道了现在只是丢失了个别报文段，于是不启动慢开始而是执行快恢复算法，调整门限值ssthresh = cwnd/2并执行拥塞避免算法。

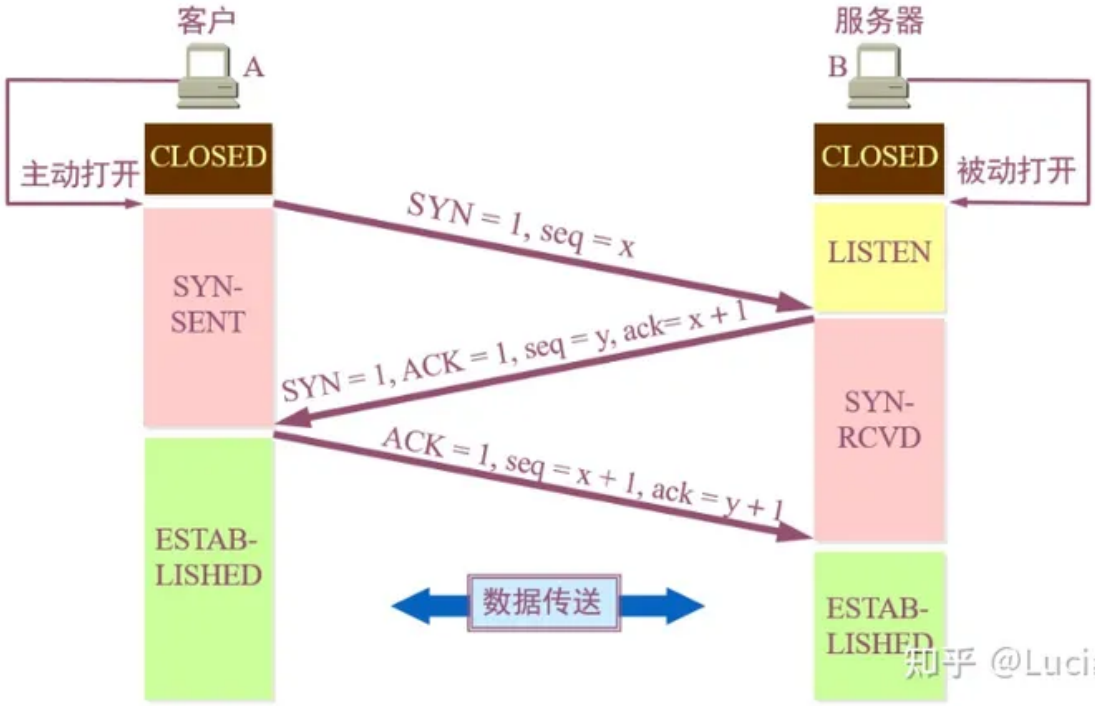




为何快重传是选择三次ACK

主要的考虑还是要区分包的丢失是由于链路故障还是乱序等其他因素引发的，两次duplicated ACK可能是乱序造成的，三次及以上duplicated ACK很可能是丢包造成的。

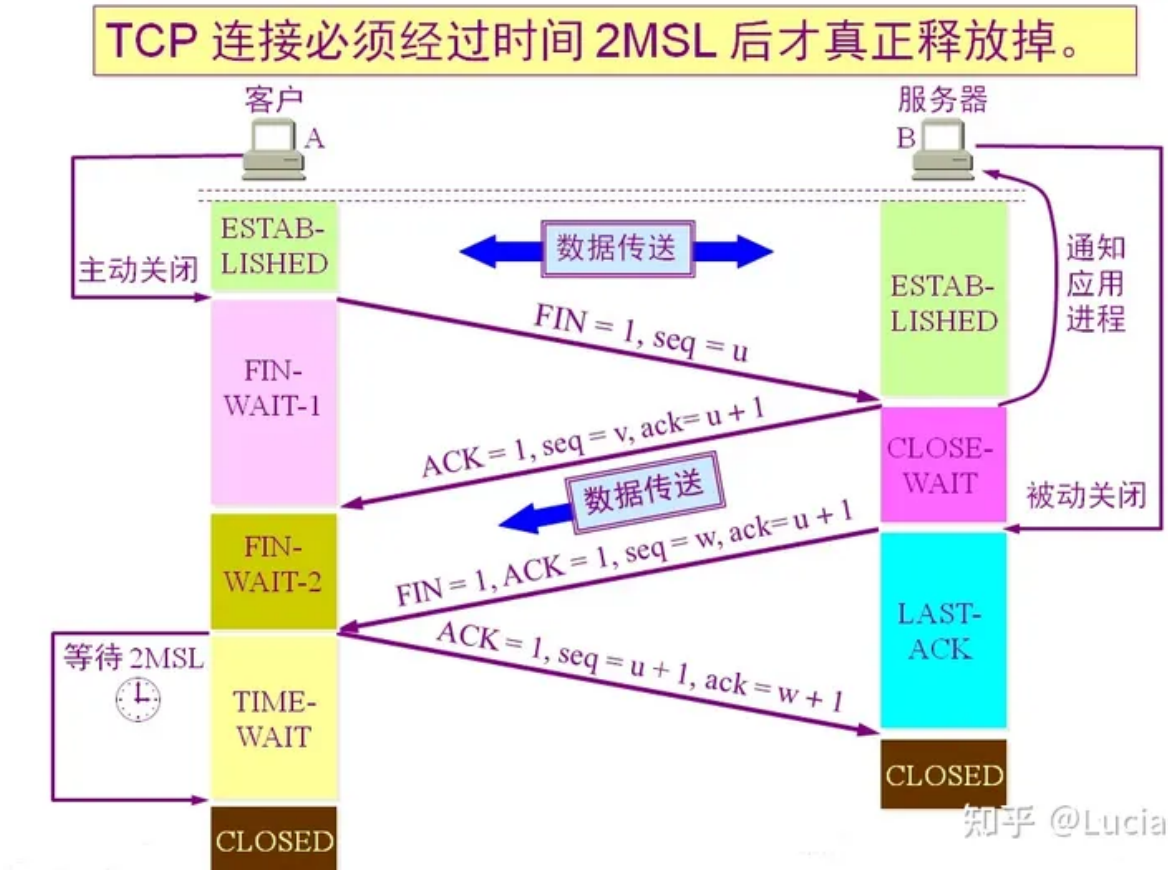
✨ TCP的三次握手和四次挥手



第一次握手：客户端向服务器端发送连接请求报文段，这时首部中的同步位SYN=1，同时选择一个初始序号seq=x，客户端进入同步已发送状态。

第二次握手：服务器端收到请求报文后，如果同意建立连接，则向客户端发送确认，在确认报文段中将同步位(SYN)和确认位(ACK)都置1，确认号是ack=x+1，同时也为自己选择一个初始序号seq=y。服务器端进入同步收到状态。

第三次握手：客户端收到服务器端的确认后，还要向服务器发送确认。确认报文段确认位为1，确认号ack=y+1，而自己的序号seq=x+1。这时，TCP连接已建立，客户端进入已建立连接状态。服务器端收到客户端的确认后也进入已建立连接状态。



第一次挥手：客户端发送连接释放报文段，首部终止位FIN=1，序号为seq=u，等于前面已传送过的数据最后一个字节的序号加1，这时客户端进入终止等待状态，等待服务器端的确认。

第二次挥手：服务器端收到连接释放报文后立即发出确认，确认号为ack=u+1，这个报文段自己的序号为seq=v，等于服务器端前面已传送过的数据最后一个字节的序号加1，服务器端进入关闭等待状态。此时TCP连接处于半关闭状态。

第三次挥手：服务器端再没有要发送的数据时，向客户端发送连接释放报文，终止位FIN=1，确认位ACK=1，序号为seq=w，确认号为ack=u+1，此时服务器进入最后确认状态，等待客户端确认。

第四次挥手：客户端收到服务器端的连接释放报文后，对此发出确认。确认位ACK=1，确认号ack=w+1，自己的序号为seq=u+1，然后进入时间等待状态。

✨ 为什么需要三次握手，两次不行吗？

1. 需要三次握手才能确认双方的接收与发送能力是否正常：

- 第一次握手：客户端发送网络包，服务端收到了。服务端确认客户端的发送能力以及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。客户端确认服务端的接收、发送能力以及客户端的接收、发送能力是正常的。不过此时服务端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。服务端确认客户端的接收、发送能力，服务端自己的发送、接收能力正常。
- 如果采用两次握手可能出现以下情况：客户端发出连接请求，但是连接请求报文丢失未收到确认，于是客户端重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕，释放连接。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节点长时间滞留了，延误到连接释放时候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了一次新的连接请求，于是向客户端发出确认报文段，同意建立连接，不采用三次握手，是要服务端发出确认，就建立了新的连接，此时客户端忽略服务端发出的确认，也不发送数据，则服务端一直等待客户端发送数据，浪费资源。

- 另外两次握手至多只有客户端的起始序列号能被确认，服务器端的序列号则得不到确认。

✨ 三次握手过程中可以携带数据吗?

只有第三次握手才能需携带数据。

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，那它每次在第一次握手中的SYN报文中放入大量数据。因为攻击者根本不理服务端的接收、发送能力是否正常，然后疯狂重复发送SYN报文，会让服务器花费很多时间、内存空间来接收这些报文。

✨ 为什么是四次挥手?

在三次握手过程中，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报文用来应答，SYN报文用来同步。但是连接关闭时，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，FIN报文已收到。只有等到服务端所有报文都发送完了，才能发送FIN报文，因此不能一起发送，故需要四次挥手。

✨ 四次挥手中客户端的TIME_WAIT状态

客户端收到服务器端的FIN报文之后进入此状态，此时并不是直接进入CLOSED状态，还需要等待一个时间计时器设置的时间`2*MSL`（MSL报文最大生存时间）。这样做有两个理由：

1. 确保最后一个确认报文能够到达；
2. 等待一段时间是为了让本连接持续时间内所产生的所有报文都从网络中消失，使得下一个新的连接不会出现旧的连接请求报文。

✨ TCP协议如何保证可靠传输

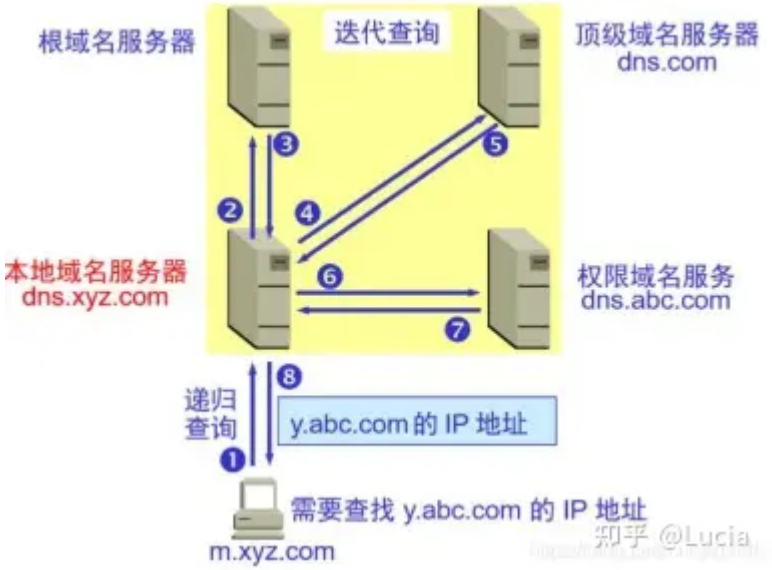
- **确认和重传**：接收方收到报文就会确认，发送方发送一段时间后没有收到确认就会重传。
- **数据校验**：TCP报文头有校验和，用于校验报文是否损坏。
- **数据合理分片和排序**：TCP会按最大传输单元合理分片，接收方会缓存未按序到达的数据，重新排序后交给应用层，而UDP：IP数据报大于1500字节，大于MTU。这个时候发送方的IP层就需要分片，把数据报分成若干片，使得每一篇都小于MTU。而接收方IP层则需要进行数据报的重组。由于UDP的特性，某一片数据丢失时，接收方便无法重组数据报，导致丢弃整个UDP数据报。
- **流量控制**：当接收方来不及处理发送方的数据，能通过滑动窗口，提示发送方降低发送的速率，防止包丢失。
- **拥塞控制**：当网络拥塞时，通过拥塞窗口减少数据的发送，防止包丢失。

✨ DNS域名解析

DNS（Domain Name System, 域名系统），因特网上作为域名和IP地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的IP数串。通过主机名，最终得到该主机名对应的IP地址的过程叫做域名解析。

DNS缓存

为了提高DNS查询效率，并减轻根域名服务器的负荷和减少互联网上的DNS查询报文数量，在域名服务器中广泛使用了高速缓存，用来存放最近查询过的域名以及从何处获得域名映射信息的记录。（本地域名服务器和主机上）



工作原理

将主机域名转换为IP地址，属于应用层协议，使用UDP协议。

1. **主机向本地域名服务器的查询一般采用递归查询：**如果主机所查询的本地域名服务器不知道被查询的IP地址，那么本地域名服务器就以DNS客户的身份，向其他根域名服务器继续发出查询请求报文。（替该主机进行查询）
2. **本地域名服务器向根域名服务器的查询通常是迭代查询：**当根域名服务器收到本地域名服务器发出的迭代请求报文时，要么给出所要查询的IP地址，要么告诉本地域名服务器下一步应该向哪一个域名服务器进行查询。（不是替本地域名服务器进行后续查询）

DNS负载均衡

当一个网站有足够多的用户的时候，假如每次请求的资源都位于同一台机器上面，那么这台机器随时可能会崩掉。处理办法就是用DNS负载均衡技术，它的原理是在DNS服务器中为同一个主机名配置多个IP地址，在应答DNS查询时，DNS服务器对每个查询将以DNS文件中主机记录的IP地址按顺序返回不同的结果，将客户端的访问引导到不同的机器上去，使得不同的客户端访问不同的服务器，从而达到负载均衡的目的。例如可以根据每台机器的负载量，该机器离用户地理位置的距离等等。

✨文件传送协议FTP

FTP屏蔽了各计算机系统的细节，适合于在异构网络中任意计算机之间传送文件，是因特网上使用的最广泛的文件传输协议。

FTP使用**客户服务器方式**，一个FTP服务器进程可同时为多个用户进程提供服务。FTP的服务器进程由两大部分组成：一个**主进程**，负责接收新的请求，另外有若干**从属进程**，负责处理单个请求。

FTP使用两个TCP连接：

- **控制连接**在整个会话期间一直保持打开，FTP客户发出的传送请求通过控制连接发送给服务器端的控制进程，但控制连接不用来传送文件。
- **数据连接**和数据传送进程实际完成文件的传送，在传送完毕后关闭"数据传送连接"并结束运行。

当客户进程向服务器进程发出建立连接请求时，要寻找连接服务器进程的熟知端口(21)， 同时还要告诉服务器进程自己的另一个端口号码， 用于建立数据传送连接。

接着，服务器进程用自己传送数据的熟知端口(20)与客户进程所提供的端口号码建立数据传送连接。

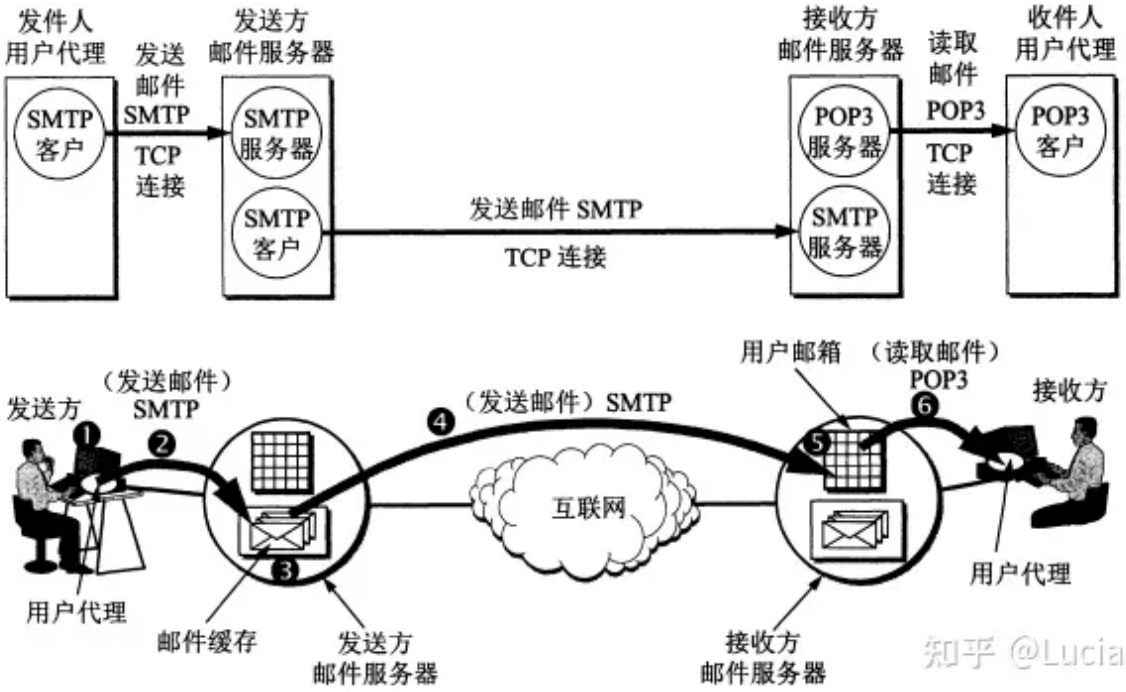
由于FTP使用了两个不同的端口号， 所以数据连接与控制连接不会发生混乱。

✨电子邮件

发送邮件的协议： SMTP

读取邮件的协议： POP3 和 IMAP

发送和接收电子邮件的几个重要步骤：



1. 发件人调用PC中的用户代理撰写和编辑要发送的邮件。
2. 发件人的用户代理把邮件用SMTP协议发送给发送方邮件服务器。
3. SMTP服务器把邮件临时存放在邮件缓存队列，等待发送。
4. 发送方邮件服务器的SMTP客户与接收方邮件服务器的SMTP服务器建立TCP连接，然后就把邮件缓存队列中的邮件依次发送出去。
5. 运行在接收方邮件服务器中的SMTP服务器进程收到邮件后，把邮件放入收件人的用户邮箱中，等待收件人进行读取。
6. 收件人在打算收信时，就运行PC机中的用户代理，使用POP3（或IMAP）协议读取发送给自己的邮件。

✨ HTTP协议请求方法

请求方法	说明
GET	请求服务器某一资源
POST	向指定资源提交数据，数据包含在请求体中
HEAD	类似GET请求，只不过返回的响应中没有具体的内容，用于获取报头
PUT	通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定资源的内容
DELETE	请求服务器删除指定的资源
CONNECT	当浏览器配置为使用代理服务器时才会用到CONNECT方法，建立隧道，实现用隧道协议进行TCP通信

请求方法	说明
OPTIONS	用于描述目标资源的通信选项，查看服务器支持哪些请求方法
TRACE	用于追踪路径，发送请求时，首部字段Max-Forwards会指定一个数值，每经过一个服务器之后，该数值减1，当该数值为0时，停止传输，最后接收到服务器响应。
PATCH	对已知资源进行局部更新

✨HTTP报文结构

请求报文的报文结构

请求行：请求方法 + 请求URL + HTTP版本
请求头：字段名和对应的值
空行：请求头之后是一个空行，主要是告知服务器下面不再有请求头。
请求体：由用户自定义添加，例如post方法的body部分。

响应报文的报文结构

状态行： HTTP版本 + 状态码 + 状态描述符（OK）
响应头：字段名 + 对应的值
空行
响应体

✨HTTP响应码

状态码	状态信息	说明
1**		信息提示，需要请求者继续执行操作
100	Continue	请求中间状态，客户端需要继续下一步请求
101	Switching Protocol	协议切换，例如当HTTP请求升级到websocket服务的时候，如果服务端同意升级，返回101状态码
2**		成功
200	OK	请求成功
201	Created	请求成功并创建了新的资源
202	Accepted	服务端已接受请求，但未完成处理
204	No Content	没有内容，只返回head头信息，主体内容没返回
206	Partial Content	表示部分内容，常见于大型文件上传或下载过程
3**		重定向
301	Moved Permanently	永久重定向，将资源永久移到新的域名下
302	Found	临时重定向，将资源临时移到新的地方，以后还有可能再移回来

状态码	状态信息	说明
304	Not Modified	资源没修改，用之前缓存就行
305	Use Proxy	使用代理，表示当前请求的资源必须通过代理访问
4**		客户端错误
401	Unauthorized	要求请求方，也就是用户需要进行身份认证
403	Forbidden	服务器拒绝访问，因法律、敏感词汇等原因，服务器拒绝客户端的请求
404	Not Found	资源未找到
405	Method Not Allowed	请求方法错误
408	Request Timeout	客户端发送请求时间超时，服务器没有时间继续等待了
5**		服务器错误
501	Not Implemented	表示服务器不支持客户端的请求功能
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
503	Service Unavailable	服务器内部较忙，暂时无法响应
504	Gateway Timeout	充当网关或代理的服务器，未及时从远端服务器获取请求

301和302状态码

- 301 代表访问的地址的资源被永久移除了，以后都不应该访问这个地址，搜索引擎抓取的时候也会用新的地址替换这个老的。可以在返回的响应的 location 首部去获取到返回的地址。301的场景：域名跳转，比如从`http://baidu.com`，跳转到`https://baidun.com`，域名换了。
- 302 表示临时重定向，这个资源只是暂时不能被访问了，但是之后过一段时间还是可以继续访问，302的场景：临时跳转，比如未登陆的用户访问用户中心重定向到登录页面。
- 302不安全，尽量使用301跳转，因为302会引起网址劫持。解释：某个人在他自己的网址上做了一个302跳转，(redirect)重定向到一家知名网址，处于某种原因，搜索引擎仍然收录了他自己的网址，但是所展示的网页内容却是知名网址的内容。在不知不觉中，知名网站就给他自己的网站做贡献，然后他自己的网站排名就靠前了。这种情况就是网址URL劫持。

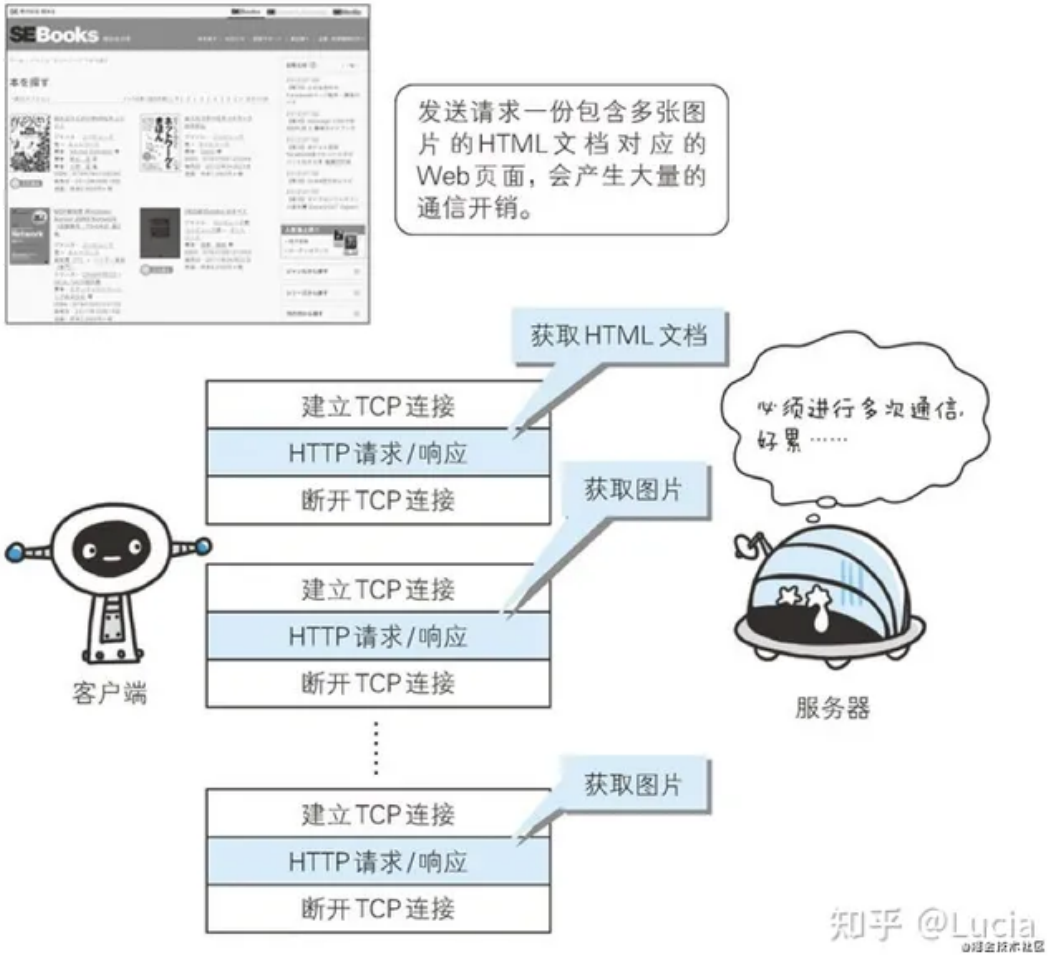
✨HTTP 1.0、HTTP1.1、HTTP2.0

HTTP 1.0

HTTP 1.0是一种无状态、无连接的应用层协议。

1. **无连接**：HTTP1.0规定浏览器和服务器保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器处理完成后立即断开TCP连接。

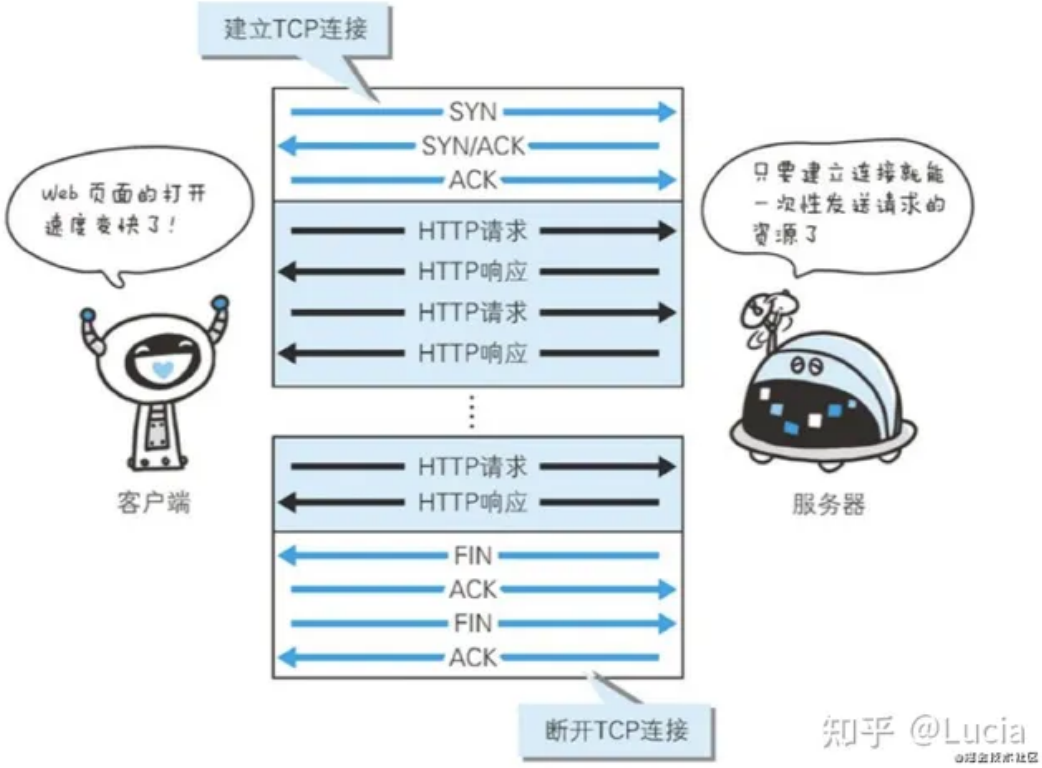
- 首先，无连接的特性导致最大的性能缺陷就是无法复用连接。每次发送请求的时候，都需要进行一次TCP的连接，而TCP的连接释放过程又是比较费事的。这种无连接的特性会使得网络的利用率非常低。
- 其次就是队头阻塞（head of line blocking）。由于HTTP1.0规定下一个请求必须在前一个请求响应到达之后才能发送。假设前一个请求响应一直不到达，那么下一个请求就不发送，同样的后面的请求也给阻塞了。



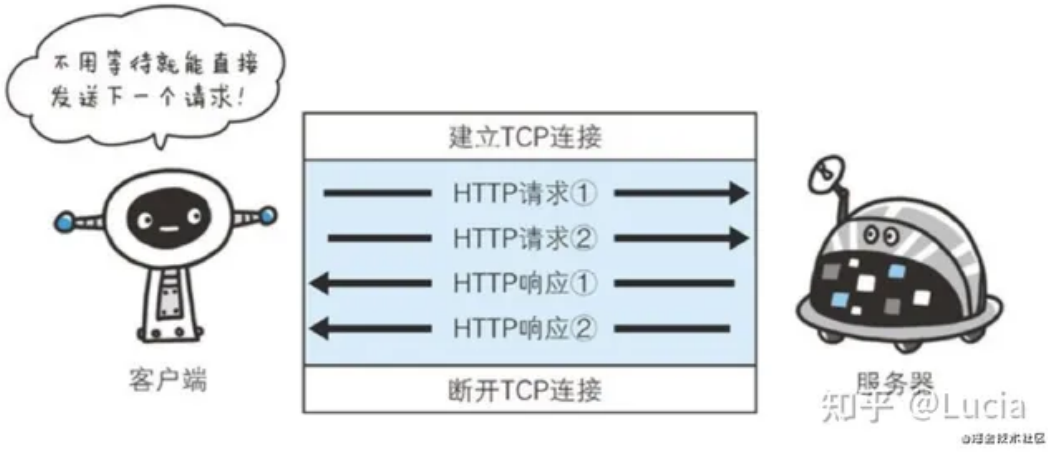
2. **无状态**：服务器不跟踪每个客户端也不记录过去的请求。 这种无状态性可以借助`cookie/session`机制来做身份认证和状态记录。

HTTP 1.1

1. **长连接**：HTTP1.1增加了一个Connection字段，通过设置Keep-Alive可以保持HTTP连接不断开，避免了每次客户端与服务器请求都要重复建立释放建立TCP连接，提高了网络的利用率。如果客户端想关闭HTTP连接，可以在请求头中携带Connection: false来告知服务器关闭请求。



2. **管道化**：基于HTTP1.1的长连接，使得请求管线化成为可能。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。但服务器必须按照客户端请求的先后顺序依次回送相应的结果，以保证客户端能够区分出每次请求的响应内容。

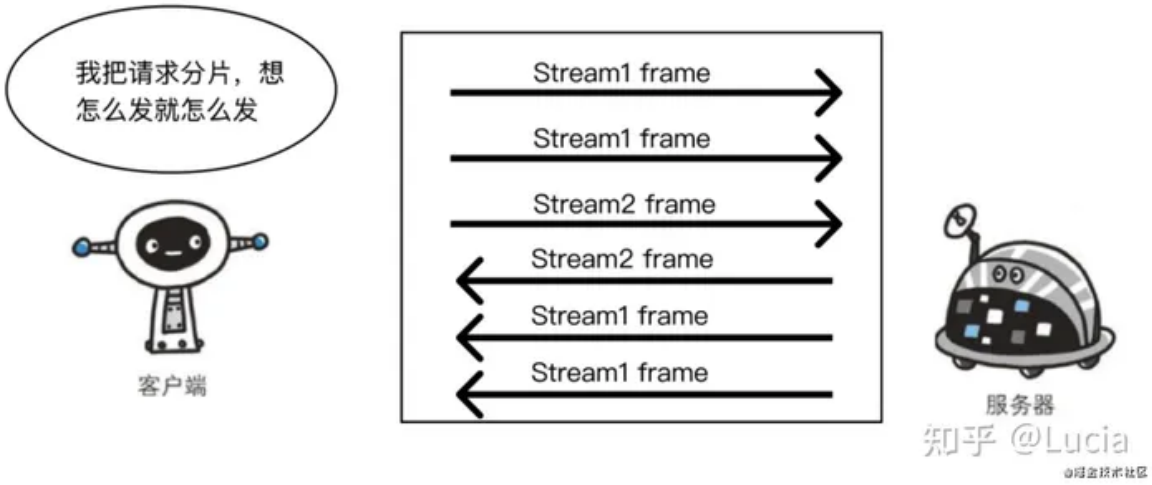


3. **增加缓存处理：**HTTP/1.0 提出缓存概念，即强缓存 Expires 和协商缓存 Last-Modified。后 HTTP/1.1 又有了更好的方案，即强缓存 Cache-Control 和协商缓存 ETag。

Expires 即过期时间，但问题是这个时间点是服务器的时间，如果客户端的时间和服务器时间有差，就不准确。所以用 Cache-Control 来代替，它表示过期时长，这就没歧义了。
Last-Modified 即最后修改时间，而它能感知的单位时间是秒，也就是说如果在1秒内改变多次，内容文件虽然改变了，但展示还是之前的，存在不准确的场景，所以就有了 ETag，通过内容给资源打标识来判断资源是否变化。

HTTP 2.0

1. **二进制分帧**：二进制分帧层通过将所有传输的信息分割为更小的消息和帧，并采用二进制格式进行编码，其中，HTTP1.1中的首部信息header封装到Headers帧中，而request body则封装到Data帧中。
2. **多路复用（连接共享）**：由于 HTTP 1.X 是基于文本的，因为是文本，就导致了它必须是个整体，在传输是不可切割的，只能整体去传。但 HTTP 2.0 是基于二进制流的。将 HTTP 消息分解为独立的帧，交错发送，然后在另一端重新组装。并行交错地发送多个请求，请求之间互不影响。并行交错地发送多个响应，响应之间互不干扰。使用一个连接并行发送多个请求和响应。简单的来说： 在同一个TCP连接中，同一时刻可以发送多个请求和响应，且不用按照顺序一一对应。之前是同一个连接只能用一次， 如果开启了keep-alive，虽然可以用多次，但是同一时刻只能有一个HTTP请求。(帧代表着最小的数据单位，每个帧会标识出该帧属于哪个流。流就是多个帧组成的数据流。)



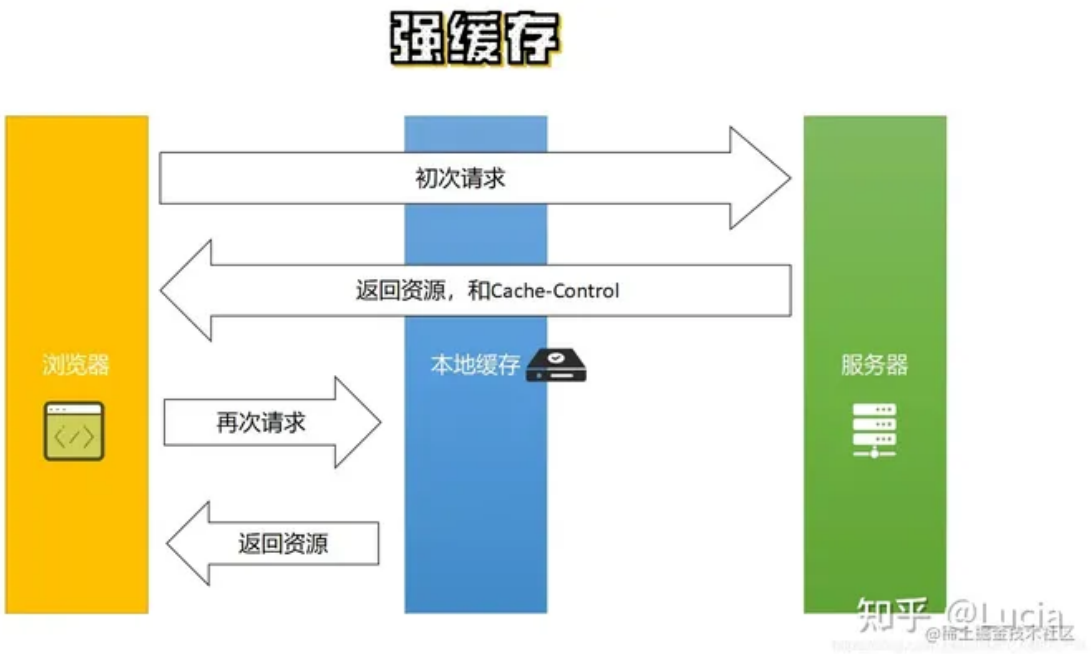
3. **头部压缩**：HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。高效的压缩算法可以很大的压缩header，减少发送包的数量从而降低延迟。
4. **服务器推送**：服务器除了对最初请求的响应外，服务器还可以额外的向客户端推送资源，而无需客户端明确的请求。

✨ HTTP缓存

缓存是一种保存资源副本并在下次请求时直接使用该副本的技术。缓存的目的其实就是为了尽量减少网络请求的体积和数量，让页面加载的更快。静态资源（js、css、img）可以被缓存。

强制缓存

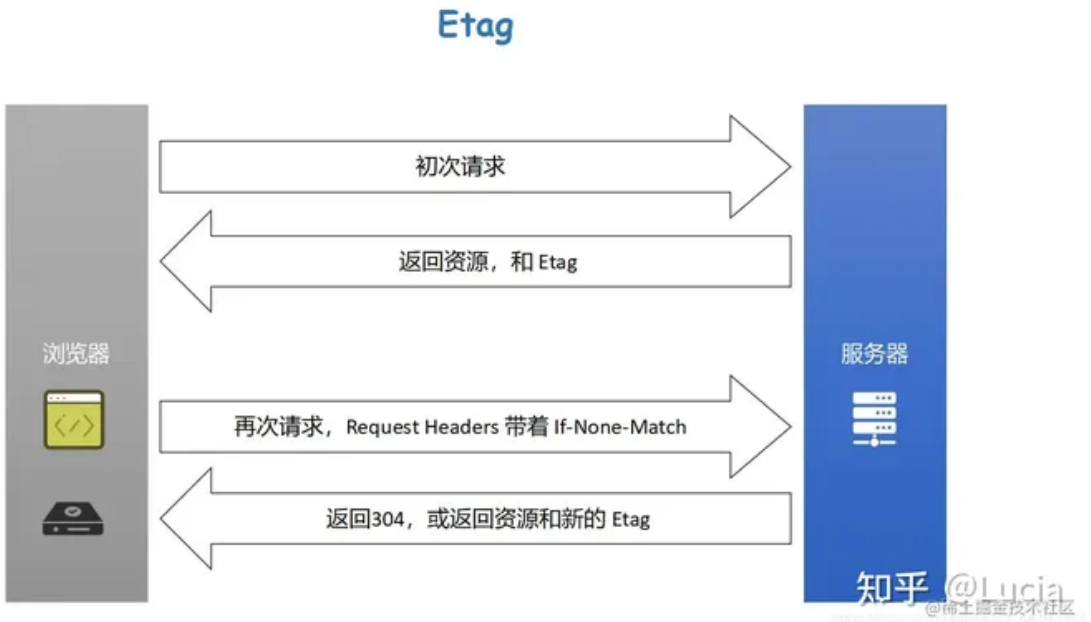
强制缓存就是文件直接从本地缓存中获取，不需要发送请求。 当初次请求时，浏览器会向服务器发起请求，服务器接收到浏览器的请求后，返回资源并返回一个 Cache-Control 给客户端，该 Cache-Control（max-age）一般设置缓存的最大过期时间。浏览器已经接收到 cache-control 的值，那么这个时候浏览器再次发送请求时，它会先检查它的 cache-control 是否过期，如果没有过期则直接从本地缓存中拉取资源，返回到客户端，而无需再经过服务器。如果客户端的 cache-control 失效了，需要重新向服务器发起请求，之后服务器会再次返回资源和 cache-control 的值。



协商缓存

协商缓存，也叫对比缓存。它是一种服务端的缓存策略，即通过服务端来判断某件事情是不是可以被缓存。服务端判断客户端的资源，是否和服务端资源一样，如果一致则返回304 ，反之返回200和最新的资源。每次都要向服务器发送请求。 首先，如果客户端是第一次向服务器发出请求，则服务器返回资源和相对应的资源标识给浏览器。该资源标识就是对当前所返回资源的一种唯一标识，可以是 Etag 或者是 Last-Modified。当响应头部 Response Headers 同时存在 Last-Modified 和 Etag 的值时，会优先使用 Etag。

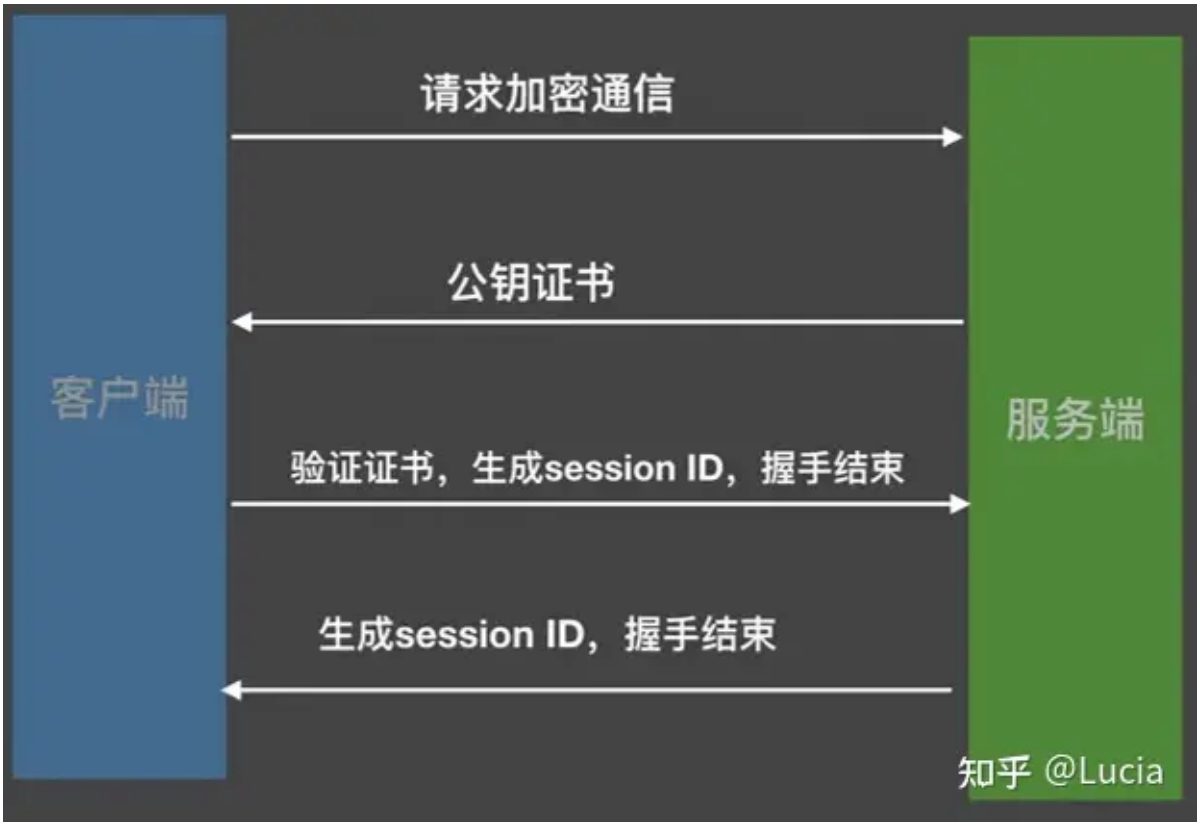
当浏览器第一次发送请求时，服务器返回资源并返回一个 Etag 的值给浏览器。这个 Etag 的值给到浏览器之后，浏览器会通过 If-None-Match 的字段来保存 Etag 的值，且 If-None-Match 保存在请求头当中。之后当浏览器再次发送请求时，请求头会带着 If-None-Match 的值去找服务器，服务器此刻就会匹配浏览器发过来的 If-None-Match 是否和自己最后一次修改的 Etag 的值相等。如果相等，则返回 304 ，表示资源未被修改；如果不相等，则返回 200 ，并返回资源和新的 Etag 的值。



✨HTTPS

HTTPS是超文本传输安全协议，加强版的HTTP，在HTTP的基础上对headers、body进行加密后再传输，大大增加了安全性。简单的讲，HTTPS = HTTP + SSL/TLS。

- **SSL/TLS（安全套接字层）**：是一套复杂的保证通信安全的协议，位于应用层和TCP/IP层之间。用HTTP协议通信的双方只需要建立TCP连接后就可以直接通信，而应用了SSL/TLS协议的通信双方还需要建立SSL连接，再进行通信。
- **SSL/TLS的大致工作流程**：客户端向服务端索要公钥并验证，双方协商生成会话密钥，通过会话密钥进行通信。
- **握手阶段**：整个HTTPS通信过程中用到非对称加密的阶段。



1. 客户端向服务器发起加密通信的请求，这次请求称为ClientHello，发送的内容包括：一个随机数x，用来后续产生会话密钥客户端支持的算法，客户端支持的SSL协议版本，支持的压缩方式。
2. 服务端收到请求后，响应客户端请求，这次请求称为ServerHello，返回如下数据：确认支持的SSL协议版本，确认加密算法，服务端公钥，为了数据安全，会放到证书中一个随机数y，用来后续生成会话密钥。
3. 客户端收到服务端响应后，首先验证证书的有效性（是否是认证机构颁发，是否在有效期内），验证通过后，会从证书中取出公钥，并向服务端发送如下数据：一个随机数z，用来后续产生会话密钥，注意这个随机数会用公钥加密，一个通知告诉服务端后续通信都用商定的加密方式和会话密钥进行，一个通知告诉服务端客户端握手结束。
4. 服务端收到最后一个随机数后，用私钥解密后就拥有了和客户端一样的三个随机数，然后生成和客户端相同的会话密钥，并做在最后的回应：一个通知告诉客户端后续通信都用商定的加密方式和会话密钥进行，一个通知告诉客户端服务端握手结束。

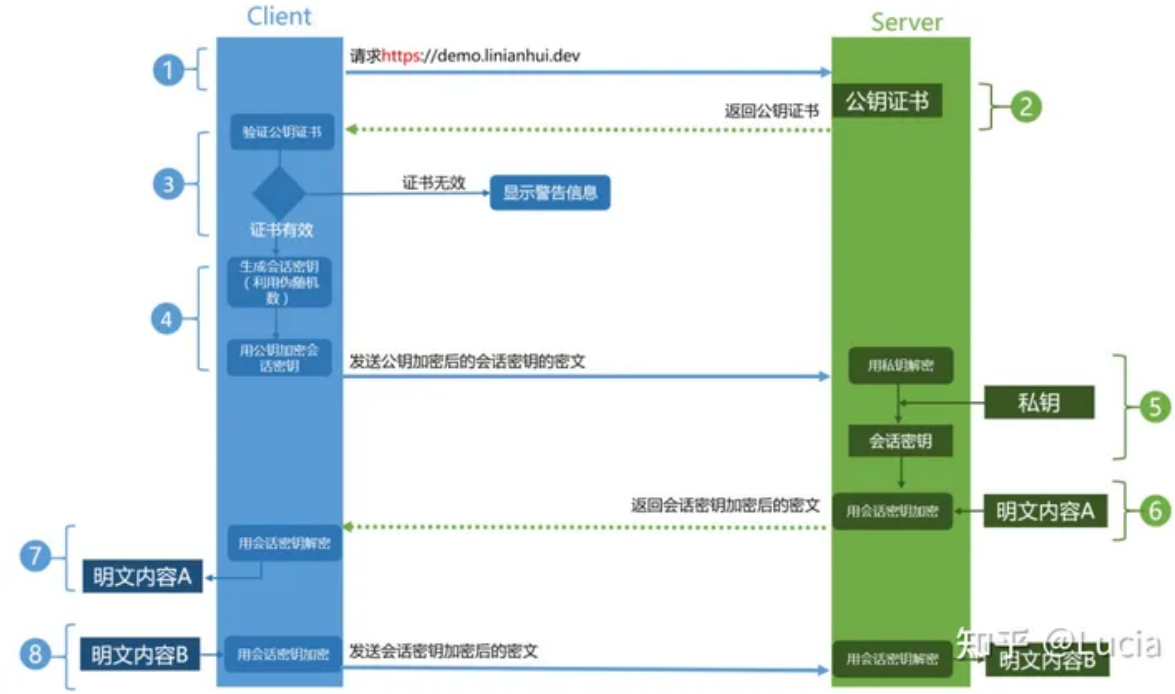
✨HTTP和HTTPS的区别

1. HTTP是明文传输，不安全的，HTTPS是加密传输，安全的多。
2. HTTP标准端口是80，HTTPS标准端口是443。

3. HTTP不用认证证书免费，HTTPS需要认证证书要钱。
4. 连接方式不同，HTTP三次握手，HTTPS中TLS1.2版本7次，TLS1.3版本6次。
5. HTTP在OSI网络模型中是在应用层，而HTTPS的TLS是在传输层。
6. HTTP是无状态的，HTTPS是有状态的。

✨HTTPS工作原理

1. 客户端向服务端发起建立HTTPS请求（客户端生成随机数R1发送给服务器，并告诉服务端自己支持哪些加密算法）；
2. 服务器端向客户端发送证书（服务端生成随机数R2，服务端公钥，从客户端支持的加密算法中选择一种双方都支持的加密算法，此算法用于之后会话密钥的生成）；
3. 客户端(SSL/TLS)解析证书验证证书合法性（是否是认证机构颁发，是否在有效期内）；
4. 证书验证通过后客户端生成会话密钥，用服务端证书的公钥加密随机数R3并发送给服务端（生成一个随机数R3使用公钥加密，利用随机数R1、R2、R3和会话密钥生成算法生成会话密钥）；
5. 服务器生成会话密钥（服务器用私钥解密客户端发过来的随机数R3，同样根据随机数R1、R2、R3和会话密钥生成算法生成会话密钥）；
6. 客户端使用会话密钥对数据进行加密，这样数据变为密文。服务器将加密后的密文发送给客户端。
7. 客户端收到服务器发送来的密文，用会话密钥对其进行对称解密，得到服务器发送的数据。



HTTPS在证书验证阶段是非对称加密，在内容传输阶段是对称加密。

- **对称加密**：加密与解密用的是同样的密钥。密钥越大，加密越强，但加密与解密的过程越慢。由于需要将密钥在网络传输，所以安全性不高。常用的对称加密算法有DES，AES等。
- **非对称加密**：非对称加密使用了一对密钥——公钥与私钥，私钥只能由一方安全保管，不能外泄，而公钥则可以发给任何请求它的人。非对称加密使用这对密钥中的一个进行加密，而解密则需要另一个密钥。安全性高，但加密与解密速度慢。常用的对称加密算法有RSA，DSA等。

✨GET和POST的区别

GET和POST是HTTP协议中的两种发送请求的方法，HTTP的底层是TCP/IP，所以GET和POST的底层也是TCP/IP。

1. 作用：GET用于获取资源，POST用于传输实体；
2. 参数：GET参数通过URL传递，POST参数放在Request Body中；
3. 安全性：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；
4. 缓存：GET请求会被浏览器主动缓存，而POST不会，除非手动设置；
5. 请求长度：GET请求在URL中传送的参数是有长度限制的，基本是2kb，而POST没有；
6. GET产生一个TCP数据包，POST产生两个TCP数据包。对于GET方式的请求，浏览器会把HTTP header和data一起发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

✨Cookie、session、localStorage、sessionStorage

Cookie和Session的区别

- **Cookie**：HTTP协议本身是无状态的，为了使其能够处理更加复杂的逻辑，HTTP/1.1引入Cookie来保存状态信息。第一次请求后，服务器返回cookie给浏览器，然后浏览器保存在本地，当用户发送第二次请求时，会自动将上次请求的cookie数据携带给服务器，服务器通过携带的数据就能判断当前用户是哪个了。
- **Session**：session和cookie都是为了存储用户相关信息，不同的是，cookie存储在本地浏览器，而session存储在服务器。session通过cookie存储一个session_id，具体的数据存储在session中，当用户再次请求服务器时，用户会通过cookie将session_id携带上来，服务器根据session_id在session库中的位置，获取用户相关的session数据。

1. **数据存放位置不同**：cookie数据存放在客户端浏览器中，session存储在服务器端；
2. **安全程度不同**：session比cookie更安全；
3. **数据存储大小不同**：单个cookie保存的数据不能超过4KB，session无限制；
4. **有效期不同**：cookie长期有效，session关闭窗口就会失效。

Cookie和webStorage的区别

1. **数据存储：**Cookie始终在同源的HTTP请求中携带，SessionStorage和localStorage不会自动把数据发送给服务器，仅在本地保存。
2. **存储数据大小：**Cookie数据不能超过4K，SessionStorage和localStorage虽然也有存储大小的限制但比Cookie大得多，可以达到5M或更大。
3. **数据存储有效期：**SessionStoage仅在当前浏览器窗口关闭之前有效；localStorage始终有效，除非手动删除，即使窗口关闭也一直保存。Cookie只在设置的Cookie过期时间之前有效，即使窗口关闭或浏览器关闭。
4. **作用域不同：**SessionStorage仅在当前页面共享，localStorage和Cookie在所有同源页面之间共享。