# 目录

# 计算机网络概述

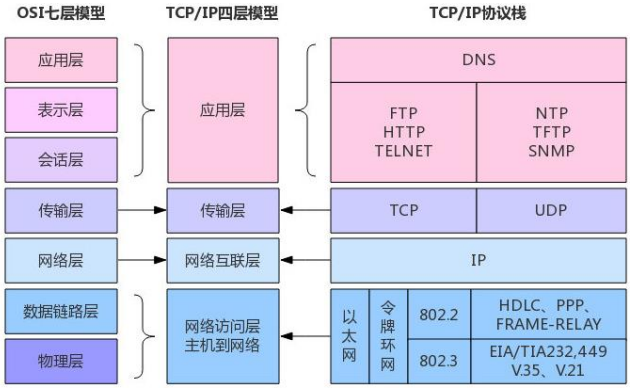
## 什么是网络协议，为什么要对网络协议分层

网络协议是计算机在通信过程中要遵循的一些约定好的规则。
网络分层的原因：

- 易于实现和维护，因为各层之间是独立的，层与层之间不会收到影响。
- 有利于标准化的制定

## 计算机网络的各层协议及作用

计算机网络体系可以大致分为一下三种，七层模型、五层模型和 TCP/IP 四层模型，一般面试能流畅回答出五层模型就可以了




| OSI中的层 | 功能 | TCP/IP协议族 |
| --- | --- | --- |
| 应用层 | 文件传输，电子邮件，文件服务，虚拟终端 | TFTP，HTTP，SNMP，FTP，SMTP，DNS，RIP，Telnet |
| 表示层 | 数据格式化，代码转换，数据加密 | 无 |
| 会话层 | 控制应用程序之间会话能力；如不同软件数据分发给不同软件 | ASAP、TLS、SSH、ISO 8327 / CCITT X.225、RPC、NetBIOS、ASP、Winsock、BSD sockets |
| 传输层 | 端到端传输数据的基本功能 | TCP、UDP |
| 网络层 | 定义IP编址，定义路由功能；如不同设备的数据转发 | IP，ICMP，RIP，OSPF，BGP，IGMP |
| 数据链路层 | 定义数据的基本格式，如何传输，如何标识 | SLIP，CSLIP，PPP，ARP，RARP，MTU |
| 物理层 | 以二进制数据形式在物理媒体上传输数据 | ISO2110，IEEE802 |

- **应用层**

应用层的任务是通过应用进程之间的交互来完成特定的网络作用，常见的应用层协议有域名系统 DNS，HTTP 协议等。

- **表示层**

表示层的主要作用是数据的表示、安全、压缩。可确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。

- **会话层**

会话层的主要作用是建立通信链接，保持会话过程通信链接的畅通，同步两个节点之间的对

话，决定通信是否被中断以及通信中断时决定从何处重新发送。。

- **传输层**

传输层的主要作用是负责向两台主机进程之间的通信提供数据传输服务。传输层的协议主要有传输控制协议 TCP 和用户数据协议 UDP。

- **网络层**

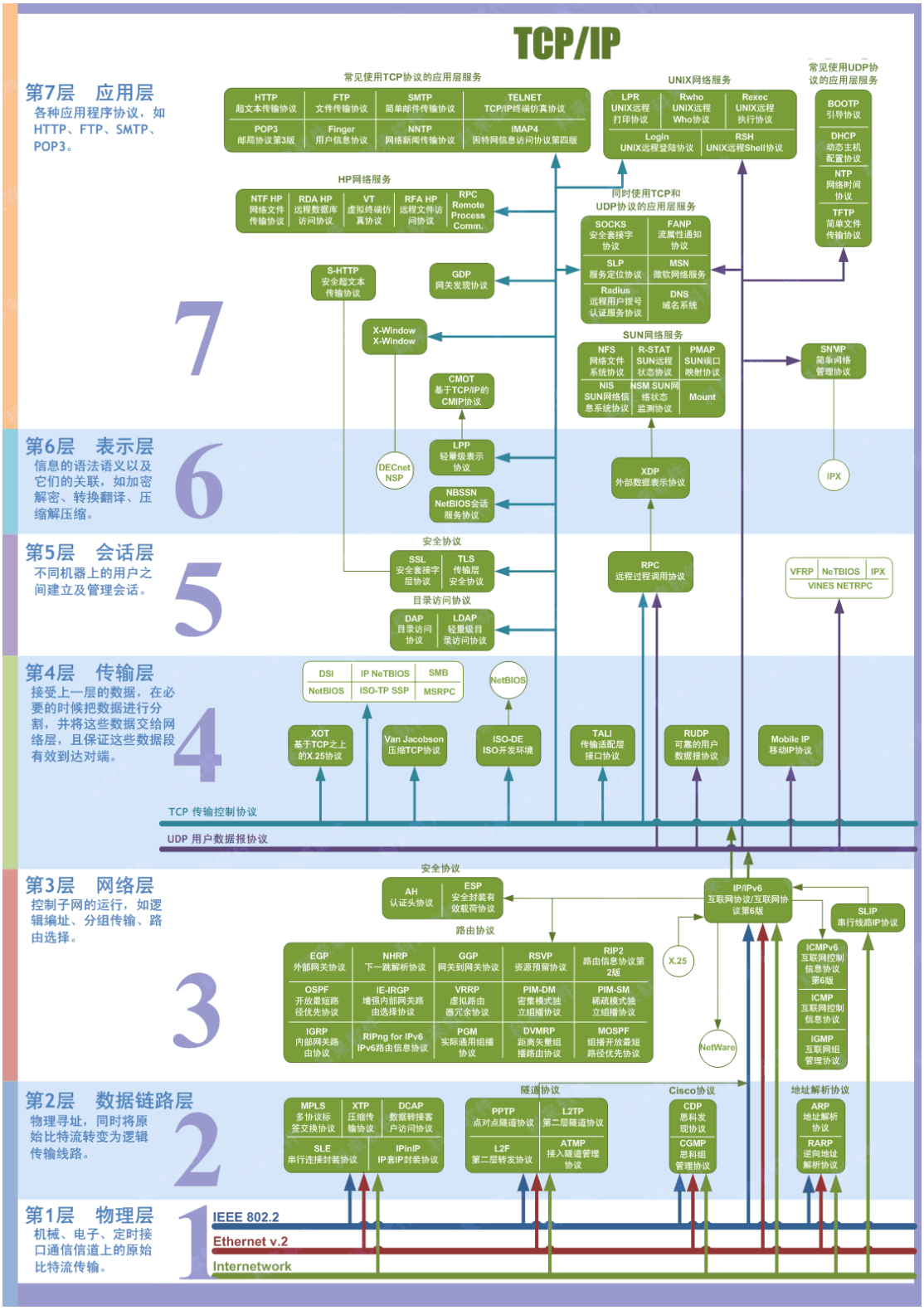
网络层的主要作用是选择合适的网间路由和交换结点，确保数据及时送达。常见的协议有 IP 协议。

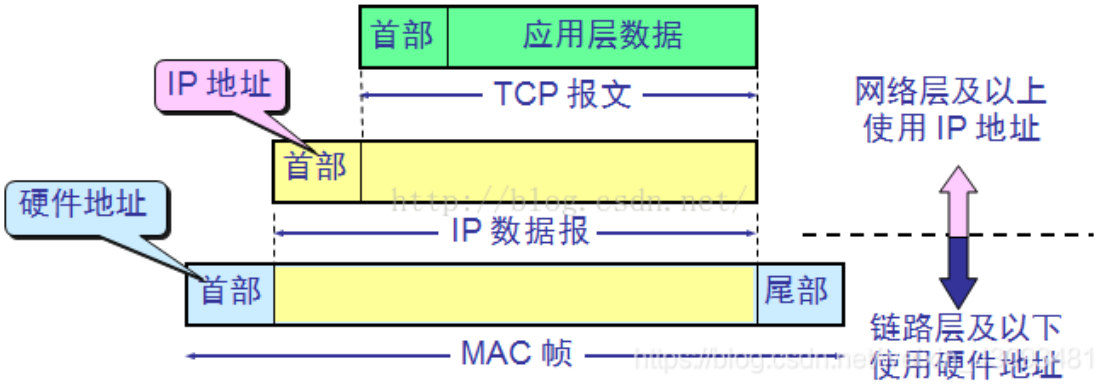
- **数据链路层**

数据链路层的作用是在物理层提供比特流服务的基础上，建立相邻结点之间的数据链路，通过差错控制提供数据帧（Frame）在信道上无差错的传输，并进行各电路上的动作系列。 常见的协议有 SDLC、HDLC、PPP 等。

- **物理层**

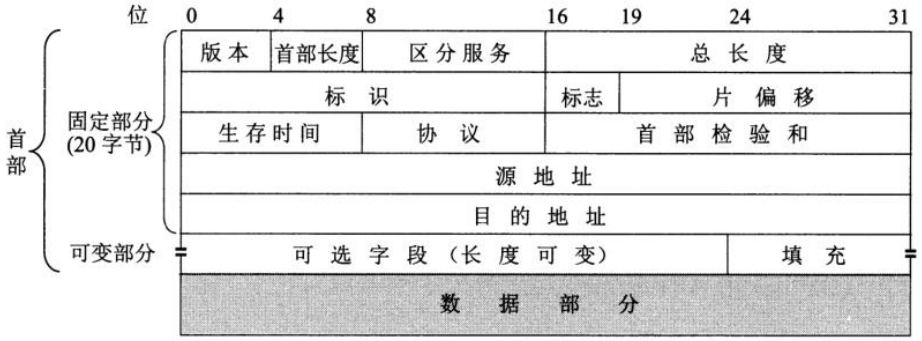
物理层的主要作用是实现相邻计算机结点之间比特流的透明传输，并尽量屏蔽掉具体传输介质和物理设备的差异。

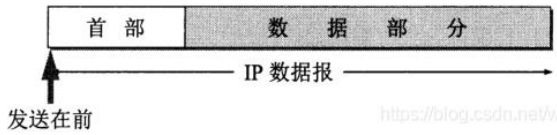

TCP/IP
第7层 应用层
各种应用程序协议，如HTTP、FTP、SMTP、POP3。
常见使用TCP协议的应用层服务
HTTP 超文本传输协议
FTP 文件传输协议
SMTP 简单邮件传输协议
TELNET TCP/IP终端仿真协议
POP3 邮局协议第3版
Finger 用户信息协议
NNTP 网络新闻传输协议
IMAP4 因特网信息访问协议第四版
UNIX网络服务
LPR UNIX远程打印协议
Rwho UNIX远程Who协议
Rexec UNIX远程执行协议
Login UNIX远程登陆协议
RSH UNIX远程Shell协议
常见使用UDP协议的应用层服务
BOOTP 引导协议
DHCP 动态主机配置协议
NTP 网络时间协议
TFTP 简单文件传输协议
HP网络服务
NTF HP 网络文件传输协议
RDA HP 远程数据库访问协议
VT 虚拟终端仿真协议
RFA HP 远程文件访问协议
RPC Remote Process Comm.
同时使用TCP和UDP协议的应用层服务
SOCKS 安全套接字协议
FANP 流属性通知协议
SLP 服务定位协议
MSN 微软网络服务
Radius 远程用户拨号认证服务协议
DNS 域名系统
S-HTTP 安全超文本传输协议
GDP 网关发现协议
X-Window X-Window
SUN网络服务
NFS 网络文件系统协议
R-STAT SUN远程状态协议
PMAP SUN端口映射协议
NIS SUN网络信息系统协议
NSM SUN网络状态监测协议
Mount
SNMP 简单网络管理协议
CMOT 基于TCP/IP的CMIP协议
7
第6层 表示层
信息的语法语义以及它们的关联，如加密解密、转换翻译、压缩解压缩。
6
DECnet NSP
LPP 轻量级表示协议
NBSSN NetBIOS会话服务协议
XDP 外部数据表示协议
IPX
第5层 会话层
不同机器上的用户之间建立及管理会话。
5
安全协议
SSL 安全套接字层协议
TLS 传输层安全协议
目录访问协议
DAP 目录访问协议
LDAP 轻量级目录访问协议
RPC 远程过程调用协议
VFRP NeTBIOS IPX
VINES NETRPC
第4层 传输层
接受上一层的数据，在必要的时候把数据进行分割，并将这些数据交给网络层，且保证这些数据段有效到达对端。
4
DSI NetBIOS
IP NeTBIOS ISO-TP SSP
SMB MSRPC
NetBIOS
XOT 基于TCP之上的X.25协议
Van Jacobson 压缩TCP协议
ISO-DE ISO开发环境
TALI 传输适配层接口协议
RUDP 可靠的用户数据报协议
Mobile IP 移动IP协议
TCP 传输控制协议
UDP 用户数据报协议
第3层 网络层
控制子网的运行，如逻辑编址、分组传输、路由选择。
3
安全协议
AH 认证头协议
ESP 安全封装有效载荷协议
IP/IPv6 互联网协议/互联网协议第6版
SLIP 串行线路IP协议
路由协议
EGP 外部网关协议
NHRP 下一跳解析协议
GGP 网关到网关协议
RSVP 资源预留协议
RIP2 路由信息协议第2版
OSPF 开放最短路径优先协议
IE-IRGP 增强内部网关路由选择协议
VRRP 虚拟路由器冗余协议
PIM-DM 密集模式独立组播协议
PIM-SM 稀疏模式独立组播协议
IGRP 内部网关路由协议
RIPng for IPv6 IPv6路由信息协议
PGM 实际通用组播协议
DVMRP 距离矢量组播路由协议
MOSPF 组播开放最短路径优先协议
X.25
NetWare
ICMPv6 互联网控制信息协议第6版
ICMP 互联网控制信息协议
IGMP 互联网组管理协议
第2层 数据链路层
物理寻址，同时将原始比特流转变为逻辑传输线路。
2
隧道协议
Cisco协议
地址解析协议
MPLS 多协议标签交换协议
XTP 压缩传输协议
DCAP 数据转接客户访问协议
SLE 串行连接封装协议
IPinIP IP套IP封装协议
PPTP 点对点隧道协议
L2TP 第二层隧道协议
L2F 第二层转发协议
ATMP 接入隧道管理协议
CDP 思科发现协议
CGMP 思科组管理协议
ARP 地址解析协议
RARP 逆向地址解析协议
第1层 物理层
机械、电子、定时接口通信信道上的原始比特流传输。
1
IEEE 802.2
Ethernet v.2
Internetwork


首部
应用层数据
IP 地址
TCP 报文
网络层及以上
使用 IP 地址
首部
硬件地址
IP 数据报
首部
尾部
链路层及以下
使用硬件地址
MAC 帧

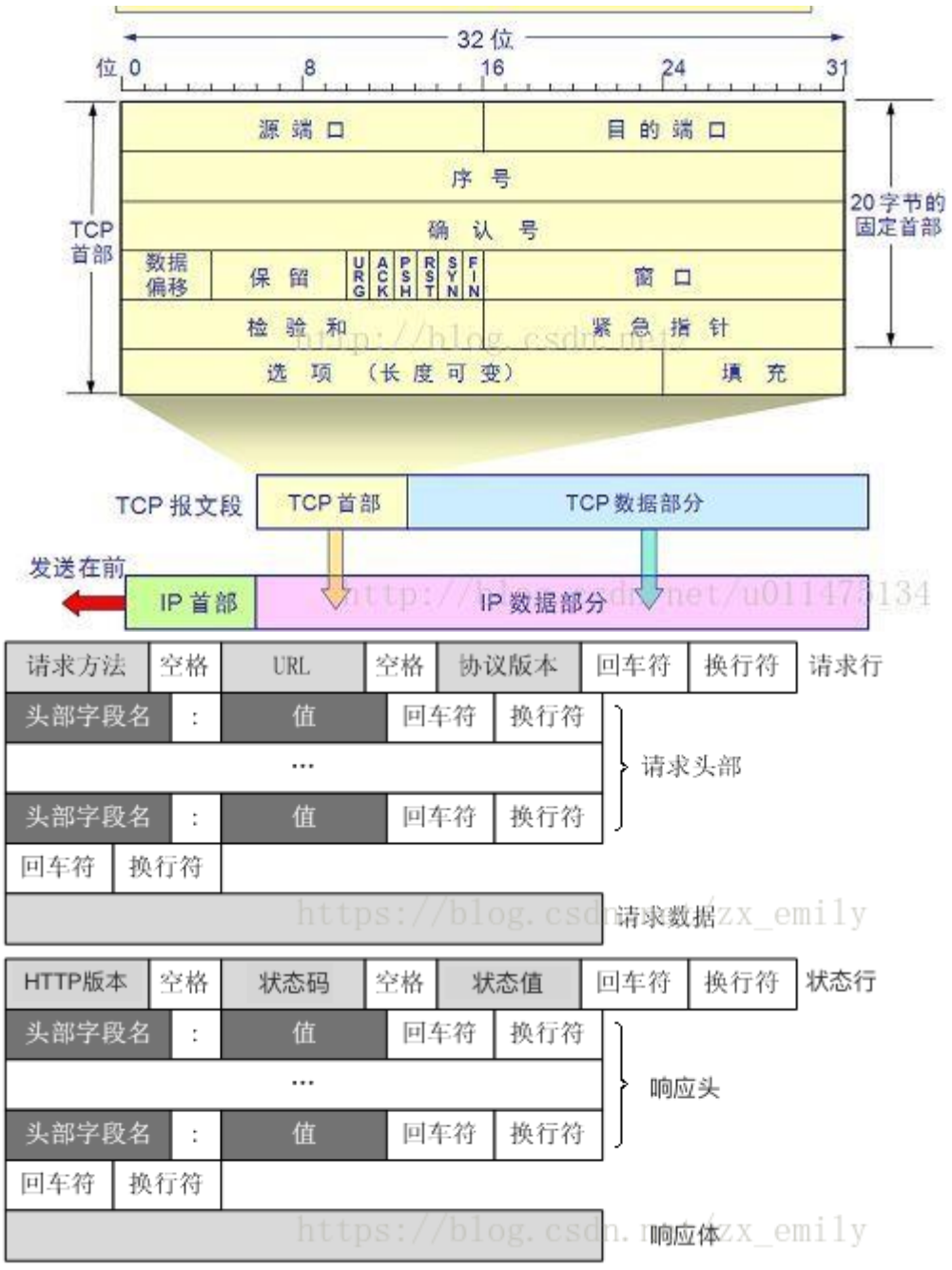
以太网帧体格式


目标MAC地址
(6字节)
源MAC地址
(6字节)
类型
(2字节)
数据
(46~1500字节)
FCS
(4字节)


位
0
4
8
16
19
24
31
首部
固定部分
(20 字节)
版本
首部长度
区分服务
总长度
标识
标志
片偏移
生存时间
协议
首部检验和
源地址
目的地址
可变部分
可选字段（长度可变）
填充
数据部分


首部
数据部分
IP 数据报
发送在前


0
15 16
31
16位源端口号
16位目的端口号
16位UDP长度
16位UDP检验和
8字节
数据(如果有)


32 位
位 0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP 首部
20 字节的固定首部
TCP 报文段
TCP 首部
TCP 数据部分
发送在前
IP 首部
IP 数据部分


请求方法
空格
URL
空格
协议版本
回车符
换行符
请求行
头部字段名
:
值
回车符
换行符
...
请求头部
回车符
换行符
请求数据
HTTP版本
状态码
状态值
状态行
响应头
响应体

# 应用层

## HTTP 协议

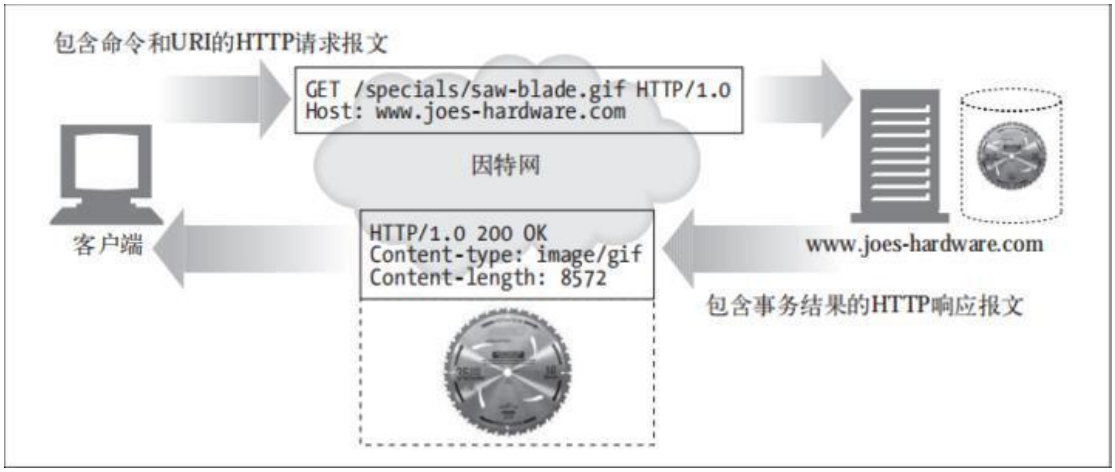
HTTP 全称是 HyperText Transfer Protocal，即：超文本传输协议。是互联网上应用最为广泛的一种网络通信协议，它允许将超文本标记语言（HTML）文档从 Web 服务器传送到客户端的浏览器。目前我们使用的是 HTTP/1.1 版本。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。

## URI 和 URL

每个 Web 服务器资源都有一个名字，这样客户端就可以说明他们感兴趣的资源是什么了，服务器资源名被称为统一资源标识符（Uniform Resource Identifier,URI）。URI 就像因特网上的邮政地址一样，在世界范围内唯一标识并定位信息资源。
统一资源定位符（URL）是资源标识符最常见的形式。 URL 描述了一台特定服务器上某资源的特定位置。
现在几乎所有的 URI 都是 URL。

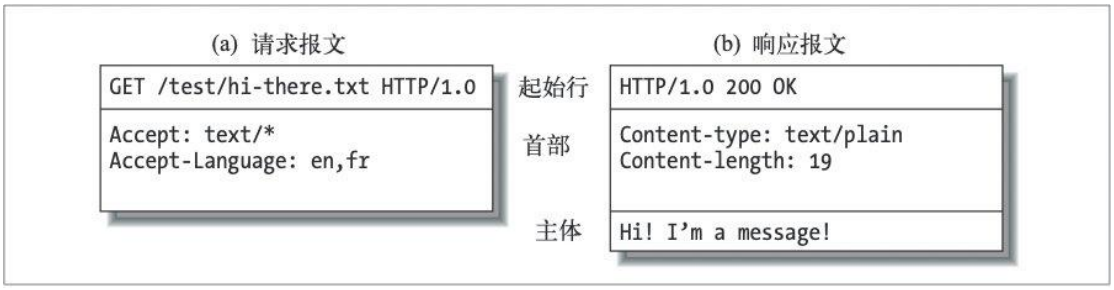
## HTTP 消息的结构

**事务和报文**
客户端是怎样通过 HTTP 与 Web 服务器及其资源进行事务处理的呢？一个 HTTP 事务由一条请求命令（从客户端发往服务器）和一个响应（从服务器发回客户端）结果组成。这种通信是通过名为 HTTP 报文（HTTP Message）的格式化数据块进行的。
**HTTP 事务：**




**报文：**

HTTP 报文是纯文本，不是二进制代码。从 Web 客户端发往 Web 服务器的 HTTP 报文称为请求报文（request message）。从服务器发往客户端的报文称为响应报文。




HTTP 报文包括三部分：

- 起始行
- 首部字段
- 主体

**HTTP 请求报文**

- 请求行：由请求方法、URL(包含参数)和协议版本组成
- 请求头部：由多个 key-value 值组成
- 空行：请求报文使用空行将请求头部和请求数据分隔
- 请求数据：GET 方法没有携带数据，POST 方法会携带一个 body


```
GET /admin_ui/rdx/core/images/close.png HTTP/1.1
Accept: */*
Referer: http://xxx.xxx.xxx.xxx/menu/neo
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CL
Accept-Encoding: gzip, deflate
Host: xxx.xxx.xxx.xxx
Connection: Keep-Alive
Cookie: startupapp=neo; is_cisco_platform=0; rdx_pagination_size=250%20Per%20Page; SESSID=deb31b8
```

**HTTP 响应报文**

- 状态行：由协议版本、状态码和状态值组成
- 响应头：由多个 key-value 值组成
- 空行：响应报文使用空行将响应头和响应体分隔
- 响应体：响应数据，在上面是一段 HTML


```
HTTP/1.1 200 OK
Bdpagetype: 1
Bdqid: 0xacbbb9d800005133
Cache-Control: private
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html
Cxy_all: baidu+f8b5e5b521b3644ef7f3455ea441c5d0
Date: Fri, 12 Oct 2018 06:36:28 GMT
Expires: Fri, 12 Oct 2018 06:36:26 GMT
Server: BWS/1.1
Set-Cookie: delPer=0; path=/; domain=.baidu.com
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=1433_21112_18560_26350_27245_22158; path=/; domain=.baidu.com
Vary: Accept-Encoding
X-Ua-Compatible: IE=Edge,chrome=1
Transfer-Encoding: chunked

<!DOCTYPE html>
<!--STATUS OK-->
```

## HTTP 请求方法

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD 方法
HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT

- GET: 通常用于请求服务器发送某些资源
- HEAD: 请求资源的头部信息，并且这些头部与 HTTP GET 方法请求时返回的一致. 该请求方法的一个使用场景是在下载一个大文件前先获取其大小再决定是否要下载，以此可以节约带宽资源
- POST: 发送数据给服务器
- OPTIONS: 用于获取目的资源所支持的通信选项
- PUT: 用于新增资源或者使用请求中的有效负载替换目标资源的表现形式
- DELETE: 用于删除指定的资源
- PATCH: 用于对资源进行部分修改

- CONNECT: HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器
- TRACE: 回显服务器收到的请求，主要用于测试或诊断

## Get 与 POST 的区别

| | get | post |
|---|---|---|
| 可见性 | 数据在 URL 中对所有人都是可见的。 | 数据不会显示在 URL 中。 |
| 安全性 | 与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 | POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。 |
| 缓存 | 能被缓存 | 不能被缓存 |
| 对数据长度的限制 | 是的。当发送数据时，GET 方法向 URL 添加数据；受浏览器对URL长度的限制的（IE的URL 的最大长度是 2048 个字符）。 | 受web服务器对post数据处理长度的限制，可设置为无限制 |
| 书签 | 可收藏为书签 | 不可收藏为书签 |

GET 与 POST 是我们常用的两种 HTTP Method，二者之间的区别主要包括如下五个方面：

- 从功能上讲，GET 一般用来从服务器上获取资源，POST 一般用来更新服务器上的资源；
- 从 REST 服务角度上说，GET 是幂等的，即读取同一个资源，总是得到相同的数据，而 POST 不是幂等的，因为每次请求对资源的改变并不是相同的；进一步地，GET 不会改变服务器上的资源，而 POST 会对服务器资源进行改变；
- 从请求参数形式上看，GET 请求的数据会附在 URL 之后，即将请求数据放置在 HTTP 报文的 请求头 中，以?分割 URL 和传输数据，参数之间以&相连。特别地，如果数据是英文字母/数字，原样发送；否则，会将其编码为 application/x-www-form-urlencoded MIME 字符串(如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密，得出如：%E4%BD%A0%E5%A5%BD，其中%XX 中的 XX 为该符号以 16 进制表示的 ASCII)；而 POST 请求会把提交的数据则放置在是 HTTP 请求报文的 请求体中。
- 就安全性而言，POST 的安全性要比 GET 的安全性高，因为 GET 请求提交的数据将明文出现在 URL 上，而且 POST 请求参数则被包装到请求体中，相对更安全。
- 从请求的大小看，GET 请求的长度受限于浏览器或服务器对 URL 长度的限制，允许发送的数据量比较小，而 POST 请求则是没有大小限制的。

HTTP 请求结构： 请求方式 + 请求 URI + 协议及其版本
HTTP 响应结构： 状态码 + 原因短语 + 协议及其版本

## PUT 和 POST 都是给服务器发送新增资源，有什么区别

PUT 和 POST 方法的区别是,PUT 方法是幂等的：连续调用一次或者多次的效果相同（无副作用），而 POST 方法是非幂等的。除此之外还有一个区别，通常情况下，PUT 的 URI 指向是具体单一资源，而 POST 可以指向资源集合。

举个例子，我们在开发一个博客系统，当我们要创建一篇文章的时候往往用 POST https://www.jianshu.com/articles，这个请求的语义是，在 articles 的资源集合下创建一篇新的文章，如果我们多次提交这个请求会创建多个文章，这是非幂等的。

而 PUT https://www.jianshu.com/articles/820357430 的语义是更新对应文章下的资源（比如修

改作者名称等），这个 URI 指向的就是单一资源，而且是幂等的，比如你把『刘德华』修改成『蔡徐坤』，提交多少次都是修改成『蔡徐坤』

## 状态码

每条 HTTP 响应报文返回时都会携带一个状态码。状态码是一个三位数字的代码，告知客户端请求是否成功，或者是都需要采取其他动作

1xx：信息状态码，表明服务端接收了客户端请求，客户端继续发送请求；

2xx：成功状态码，客户端发送的请求被服务端成功接收并成功进行了处理；

- 200 OK，表示从客户端发来的请求在服务器端被正确处理 ✨
- 201 Created 请求已经被实现，而且有一个新的资源已经依据请求的需要而建立
- 202 Accepted 请求已接受，但是还没执行，不保证完成请求
- 204 No content，表示请求成功，但响应报文不含实体的主体部分
- 206 Partial Content，进行范围请求 ✨

3xx：重定向状态码服务端给客户端返回用于重定向的信息；

- 301 moved permanently，永久性重定向，表示资源已被分配了新的 URL
- 302 found，临时性重定向，表示资源临时被分配了新的 URL ✨
- 303 see other，表示资源存在着另一个 URL，应使用 GET 方法丁香获取资源
- 304 not modified，表示服务器允许访问资源，但因发生请求未满足条件的情况

4xx：客户端错误状态码，客户端的请求有非法内容；客户端错误

- 400 Bad Request:表示请求报文中存在语法错误；
- 401 Unauthorized：经许可，需要通过 HTTP 认证；
- 403 Forbidden：服务器拒绝该次访问（访问权限出现问题）
- 404 Not Found：表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用；

5xx：服务端错误状态码，服务端未能正常处理客户端的请求而出现意外错误。

- 500 internal sever error，表示服务器端在执行请求时发生了错误 ✨
- 501 Not Implemented 请求超出服务器能力范围，例如服务器不支持当前请求所需要的某个功能，或者请求是服务器不支持的某个方法
- 503 service unavailable，表明服务器暂时处于超负载或正在停机维护无法处理请求
- 505 http version not supported 服务器不支持，或者拒绝支持在请求中使用的 HTTP 版本

## 从输入 URL 到页面展示到底发生了什么

URL 输入→DNS 解析→建立 TCP 连接→发送 HTTP 请求→服务器处理请求→服务器响应请求→浏览器解析渲染页面→连接结束

**URL 输入：**当我们开始在浏览器中输入网址的时候，浏览器其实就已经在智能的匹配可能得 url 了，他会从历史记录，书签等地方，找到已经输入的字符串可能对应的 url，然后给出智能提示，让你可以补全 url 地址。对于 google 的 chrome 的浏览器，他甚至会直接从缓存中把网页展示出来，就是说，你还没有按下 enter，页面就出来了。

**DNS 解析：**DNS 解析的过程就是寻找哪台机器上有你需要资源的过程。当你在浏览器中输入一个地址时，例如 www.baidu.com，其实不是百度网站真正意义上的地址。互联网上每一台计算机的唯一标识是它的 IP 地址，但是 IP 地址并不方便记忆。用户更喜欢用方便记忆的网址去寻找互联网上的其它计算机，也就是上面提到的百度的网址。所以互联网设计者需要在用户的方便性与可用性方面做一个权衡，这个权衡就是一个网址到 IP 地址的转换，这个过程就是 DNS 解析。它实际上充当了一个翻译的角色，实现了网址到 IP 地址的转换。

一般来说，浏览器会首先查看本地硬盘的 hosts 文件，看看其中有没有和这个域名对应的规则，如果有的话就直接使用 hosts 文件里面的 ip 地址。

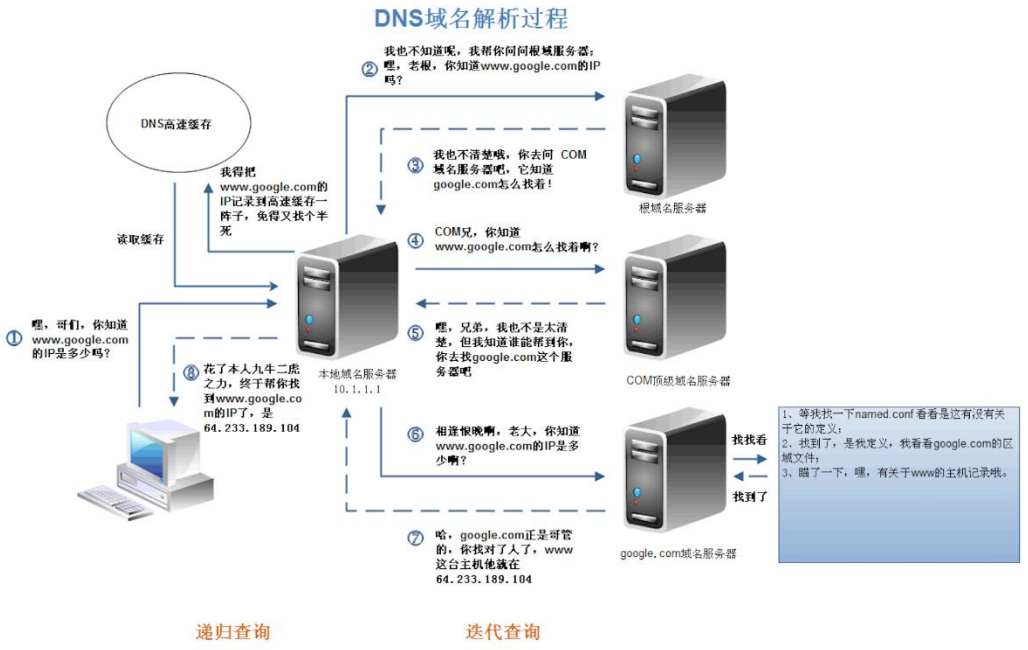
如果在本地的 hosts 文件没有能够找到对应的 ip 地址，浏览器会发出一个 DNS 请求到本地 DNS 服务器 。本地 DNS 服务器一般都是你的网络接入服务器商提供，比如中国电信，中国移动。

查询你输入的网址的 DNS 请求到达本地 DNS 服务器之后，本地 DNS 服务器会首先查询它的缓存记录，如果缓存中有此条记录，就可以直接返回结果，此过程是**递归**的方式进行查询。如果没有，本地 DNS 服务器还要向 DNS 根服务器进行查询。

根 DNS 服务器没有记录具体的域名和 IP 地址的对应关系，而是告诉本地 DNS 服务器，你可以到域服务器上去继续查询，并给出域服务器的地址。这种过程是**迭代**的过程。

本地 DNS 服务器继续向域服务器发出请求，在这个例子中，请求的对象是.com 域服务器。.com 域服务器收到请求之后，也不会直接返回域名和 IP 地址的对应关系，而是告诉本地 DNS 服务器，你的域名的解析服务器的地址。

最后，本地 DNS 服务器向域名的解析服务器发出请求，这时就能收到一个域名和 IP 地址对应关系，本地 DNS 服务器不仅要把 IP 地址返回给用户电脑，还要把这个对应关系保存在缓存中，以备下次别的用户查询时，可以直接返回结果，加快网络访问。

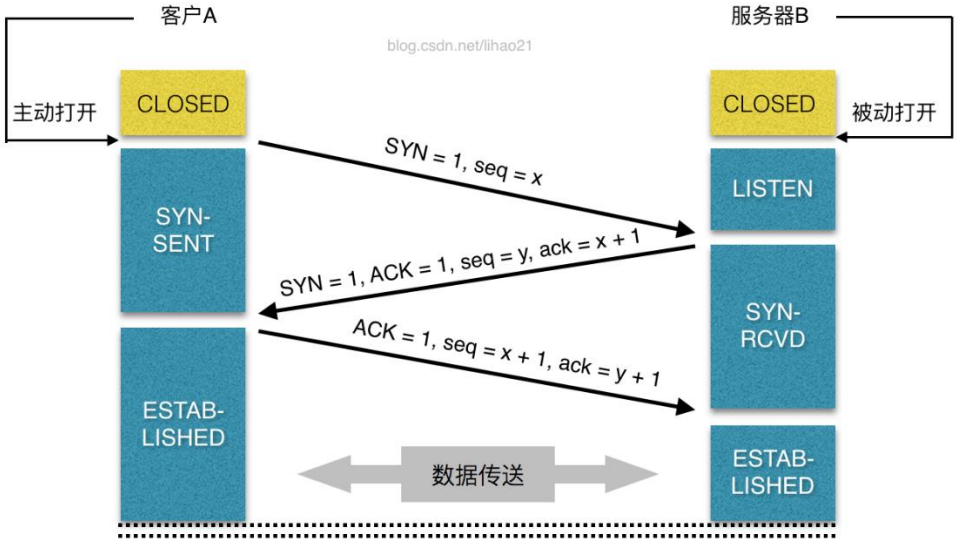



上述图片是查找 www.google.com 的 IP 地址过程。首先在本地域名服务器中查询 IP 地址，如果没有找到的情况下，本地域名服务器会向根域名服务器发送一个请求，如果根域名服务器也不存在该域名时，本地域名会向 com 顶级域名服务器发送一个请求，依次类推下去。直到最后本地域名服务器得到 google 的 IP 地址并把它缓存到本地，供下次查询使用。从上述过程中，可以看出网址的解析是一个从右向左的过程：com -> google.com -> www.google.com。但是你是否发现少了点什么，根域名服务器的解析过程呢？事实上，真正

的网址是 www.google.com.，并不是我多打了一个.，这个.对应的就是根域名服务器，默认情况下所有的网址的最后一位都是.，既然是默认情况下，为了方便用户，通常都会省略，浏览器在请求 DNS 的时候会自动加上，所有网址真正的解析过程为: . -> .com -> google.com. -> www.google.com.。

**建立 TCP 连接（这里可以穿出网络层数据链路层的知识）**

拿到域名对应的 IP 地址之后，浏览器会以一个随机端口（1024<端口<65535）向服务器的 WEB 程序（常用的有 httpd,nginx 等）80 端口发起 TCP 的连接请求。这个连接请求到达服务器端后(这中间通过各种路由设备，局域网内除外)，进入到网卡，然后是进入到内核的 TCP/IP 协议栈（用于识别该连接请求，解封包，一层一层的剥开），还有可能要经过 Netfilter 防火墙（属于内核的模块）的过滤，最终到达 WEB 程序，最终建立了 TCP/IP 的连接。




**发送 HTTP 请求：**建立了 TCP 连接之后，发起一个 http 请求。一个典型的 http request header 一般需要包括请求的方法，例如 GET 或者 POST 等，不常用的还有 PUT 和 DELETE 、HEAD、OPTION 以及 TRACE 方法，一般的浏览器只能发起 GET 或者 POST 请求。

服务器永久重定向

服务器给浏览器响应一个 301 永久重定向响应，这样浏览器就会访问 http://www.google.com/ 而非 http://google.com/。

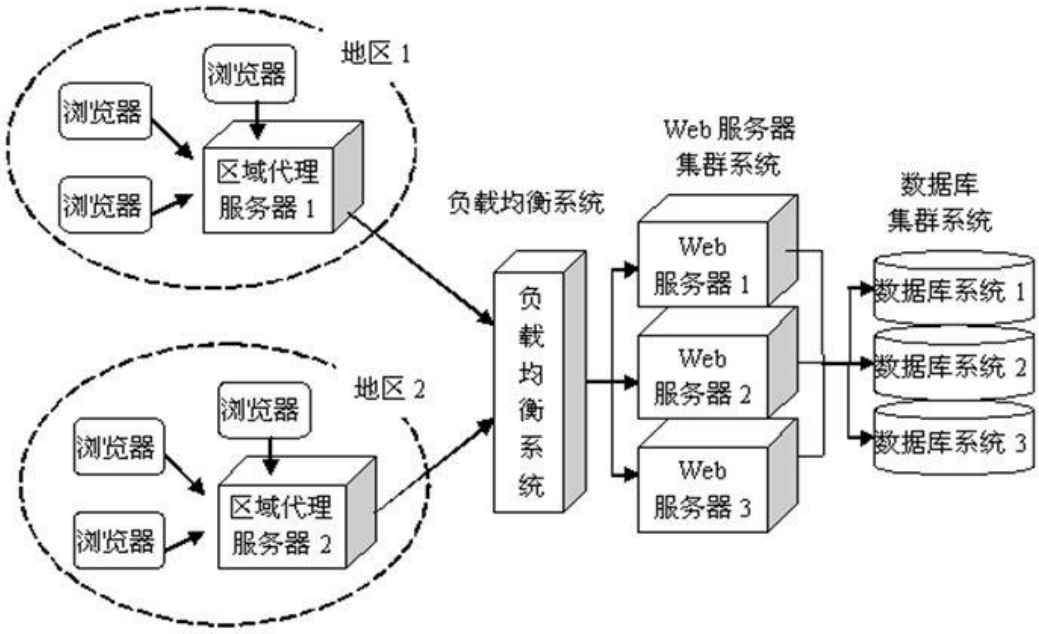
为什么服务器一定要重定向而不是直接发送用户想看的网页内容呢？其中一个原因跟搜索引擎排名有关。如果一个页面有两个地址，就像 http://www.yy.com/和 http://yy.com/，搜索引擎会认为它们是两个网站，结果造成每个搜索链接都减少从而降低排名。而搜索引擎知道 301 永久重定向是什么意思，这样就会把访问带 www 的和不带 www 的地址归到同一个网站排名下。还有就是用不同的地址会造成缓存友好性变差，当一个页面有好几个名字时，它可能会在缓存里出现好几次。

**服务器处理请求：**经过前面的重重步骤，我们终于将我们的 http 请求发送到了服务器这里，其实前面的重定向已经是到达服务器了，那么，服务器是如何处理我们的请求的呢？

后端从在固定的端口接收到 TCP 报文开始，它会对 TCP 连接进行处理，对 HTTP 协议进行解析，并按照报文格式进一步封装成 HTTP Request 对象，供上层使用。

一些大一点的网站会将你的请求到反向代理服务器中，因为当网站访问量非常大，网站越来越慢，一台服务器已经不够用了。于是将同一个应用部署在多台服务器上，将大量用户

的请求分配给多台机器处理。此时，客户端不是直接通过 HTTP 协议访问某网站应用服务器，而是先请求到 Nginx，Nginx 再请求应用服务器，然后将结果返回给客户端，这里 Nginx 的作用是反向代理服务器。同时也带来了一个好处，其中一台服务器万一挂了，只要还有其他服务器正常运行，就不会影响用户使用。




**服务器返回一个 HTTP 响应：**经过前面的 6 个步骤，服务器收到了我们的请求，也处理我们的请求，到这一步，它会把它的处理结果返回，也就是返回一个 HTPP 响应。

**浏览器显示 HTML：**浏览器在收到 HTML,CSS,JS 文件后，构建 dom 树 -> 构建 render 树 -> 布局 render 树 -> 绘制 render 树

**连接结束：**现在的页面为了优化请求的耗时，默认都会开启持久连接（keep-alive），那么一个 TCP 连接确切关闭的时机，是这个 tab 标签页关闭的时候。这个关闭的过程就是著名的四次挥手。关闭是一个全双工的过程，发包的顺序的不一定的。一般来说是客户端主动发起的关闭，过程如下。

对于一个已经建立的连接，TCP 使用改进的三次握手来释放连接（使用一个带有 FIN 附加标记的报文段）。TCP 关闭连接的步骤如下：

第一步，当主机 A 的应用程序通知 TCP 数据已经发送完毕时，TCP 向主机 B 发送一个带有 FIN 附加标记的报文段（FIN 表示英文 finish）。

第二步，主机 B 收到这个 FIN 报文段之后，并不立即用 FIN 报文段回复主机 A，而是先向主机 A 发送一个确认序号 ACK，同时通知自己相应的应用程序：对方要求关闭连接（先发送 ACK 的目的是为了防止在这段时间内，对方重传 FIN 报文段）。

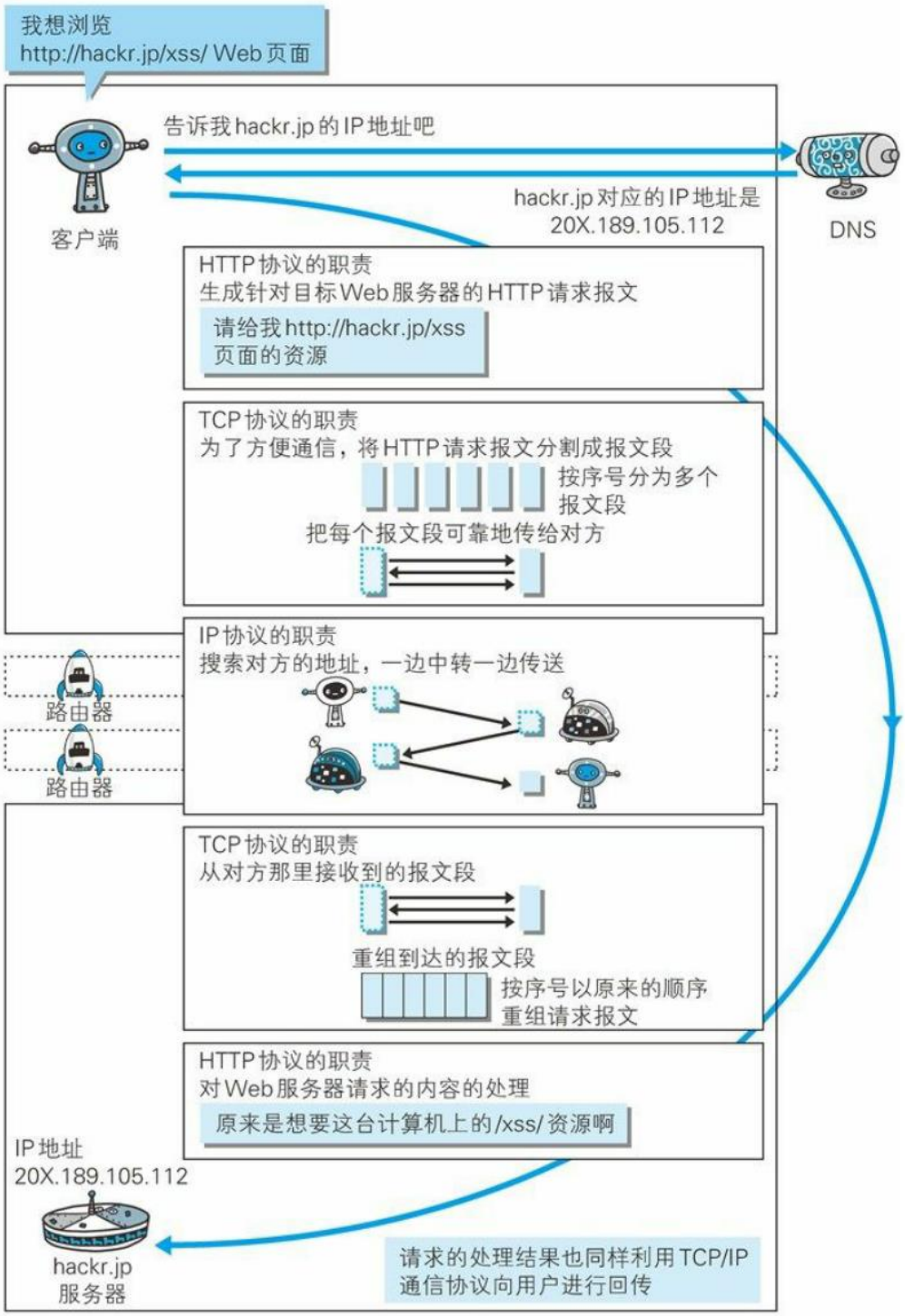
第三步，主机 B 的应用程序告诉 TCP：我要彻底的关闭连接，TCP 向主机 A 送一个 FIN 报文段。

第四步，主机 A 收到这个 FIN 报文段后，向主机 B 发送一个 ACK 表示连接彻底释放。

## HTTP 协议与各层协议之间的关系

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 层之上可靠的传递数据包，使在网络上的另一端收到发端发出的所有包，并且顺序与发出顺序一致。TCP 有可靠，面向连接的特点。




## 如何理解 HTTP 协议是无状态的

HTTP 协议是无状态的，指的是协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。也就是说，打开一个服务器上的网页和你之前打开这个服务器上的网页之间没有任何联系。HTTP 是一个无状态的面向连接的协议，无状态不代表 HTTP 不能保持 TCP 连接，更不能代表 HTTP 使用的是 UDP 协议（无连接）。

**如何实现有状态？**

- 基于 Session 实现的会话保持
  在会话开始时（客户端第一次像服务器发送 http 请求），服务器将会话状态保存起来（本机内存或数据库中），然后分配一个会话标识（SessionId）给客户端，这个会话标识一般保存在客户端 Cookie 中，以后每次浏览器发送 http 请求都会带上 Cookie 中的 SessionId 到服务器，服务器拿到会话标识就可以把之前存储在服务器端的状态信息与会话联系起来，实现会话保持（如果遇到浏览器禁用 Cookie 的情况，则可以通过 url 重写的方式将会话标识放在 url 的参数里，也可实现会话保持）
- 基于 Cookie 实现的会话保持
  基于 Cookie 实现会话保持与上述基于 Session 实现会话保持的最主要区别是前者完全将会话状态信息存储在浏览器 Cookie 中，这样一来每次浏览器发送 HTTP 请求的时候都会带上状态信息，因此也就可以实现状态保持。
- 使用隐藏域与使用 URL 重写

COOKIE 被禁用了怎么办？

最常用的就是利用 URL 重写把 Session ID 直接附加在 URL 路径的后面。

两者优缺点：

- 基于 Session 的会话保持优点是安全性较高，因为状态信息保存在服务器端。缺点是不便于服务器的水平扩展。大型网站的后台一般都不止一台服务器，可能几台甚至上百台，浏览器发送的 HTTP 请求一般要先通过负载均衡器才能到达具体的后台服务器，这就会导致每次 HTTP 请求可能落到不同的服务器上，比如说第一次 HTTP 请求落到 server1 上，第二次 HTTP 请求落到 server2 上。而 Session 默认是存储在服务器本机内存的，当多次请求落到不同的服务器上时，上述方案就不能实现会话保持了（常用解决方案是中间件，例如 Redis，将 Session 的信信息存储在 Redis 中，这样每个 server 就都可以访问到）。
- 基于 Cookie 的会话保持的优点是服务器不用保存状态信息，减轻服务端存储压力，也便于服务端做水平扩展。缺点是不够安全，因为状态信息是存储在客户端的，这意味着不能在会话中保存机密数据，另一个缺点是每次 HTTP 请求都需要发送额外的 Cookie 到服务端，会消耗更多带宽。

## HTTP 的 keep-alive 是干什么的

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。

keep-alive 的优点：

- 较少的 CPU 和内存的使用（由于同时打开的连接的减少了）
- 允许请求和应答的 HTTP 管线化
- 降低拥塞控制 （TCP 连接减少了）

- 减少了后续请求的延迟（无需再进行握手）
- 报告错误无需关闭 TCP 连

## 什么是长连接，短连接

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。

**短连接：**连接->传输数据->关闭连接

HTTP 是无状态的，浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束就中断连接。 也可以这样说：短连接是指 Socket 连接后发送后接收完数据后马上断开连接。

**长连接：**连接->传输数据->保持连接 -> 传输数据-> 。。。->关闭连接。

长连接指建立 Socket 连接后不管是否使用都保持连接。

**优点缺点？**

长连接可以省去较多的 TCP 建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在 TCP 的建立和关闭操作上浪费时间和带宽。

keep-alive 的优点：

- 较少的 CPU 和内存的使用（由于同时打开的连接的减少了）
- 允许请求和应答的 HTTP 管线化
- 降低拥塞控制 （TCP 连接减少了）
- 减少了后续请求的延迟（无需再进行握手）
- 报告错误无需关闭 TCP 连

**什么时候使用长连接什么时候使用短连接？**

长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。每个 TCP 连接都需要三步握手，这需要时间，如果每个操作都是先连接，再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，次处理时直接发送数据包就 OK 了，不用建立 TCP 连接。例如：数据库的连接用长连接， 如果用短连接频繁的通信会造成 socket 错误，而且频繁的 socket 创建也是对资源的浪费。

而像 WEB 网站的 http 服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，而像 WEB 网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。

## HTTP 各个版本的差异，0.9,1.0,1.1,2.0

**HTTP0.9**

HTTP 最早的版本是 1991 年的 0.9 版，只有一种请求方式 GET。并且服务器只能响应

HTML 格式的数据。

**HTTP/1.0**

1996 年，HTTP/1.0 发布，内容大大增加。相比于 0.9 版：

- 可以发送任意格式的数据，不仅可以传输文字，还可以传输图像、视频、二进制文件等。
- 除了 GET 请求方式，还加入了 POST、HEAD 请求方式。
- 请求和响应的格式改变，除了数据部分，每次通信必须包含头信息。
- 其他的新增功能还包括状态码（status code）、多字符集支持、多部分发送（multi-part type）、权限（authorization）、缓存（cache）、内容编码（content encoding）等。

缺点：

HTTP/1.0 的主要缺点是，每个 TCP 连接只能发送一个请求，发送数据完毕，连接就关闭了，如果还要请求其他资源，就必须重新建立连接。

TCP 建立连接成本很高，因为需要客户端和服务器三次握手，并且开始时发送速率较慢（慢启动）。所以，HTTP 1.0 版本的性能比较差。随着网页加载的外部资源越来越多，这个问题就愈发突出了。为了解决这个问题，有些浏览器在请求时，用了一个非标准的 Connection 字段。Connection: keep-alive，这个字段要求服务器不要关闭 TCP 连接，以便其他请求复用，服务器同样响应这个字段。这样一个可以复用的 TCP 连接就建立了，直到客户端或服务器主动关闭连接。

**HTTP/1.1**

1997 年，HTTP/1.1 版本发布，它进一步完善了 HTTP 协议，直到现在还是最流行的版本。相比于 1.0 版：

- HTTP1.1 最大变化是默认支持长连接。即 TCP 连接默认不关闭，可以被多个请求复用，不用声明 Connection: keep-alive。
- 管道机制，即在同一个 TCP 连接中，客户端可以同时发送多个请求，进一步改进了 HTTP 协议的效率。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。
- 增加了 PUT、PATCH、OPTIONS、DELETE 等请求方式。
- 客户端请求的头信息增加了 Host 字段，用来指定服务器的域名。Host:www.example.com。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，为虚拟主机的兴起打下了基础。（一台服务器只有一个 IP 地址，但可以有多个域名对应）

**缺点：**

虽然 1.1 版允许复用 TCP 连接，但是同一个 TCP 连接里面，所有的数据通信是按次序进行的。服务器只有处理完一个回应，才会进行下一个回应。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」（Head-of-line blocking）。

**HTTP/2.0**

HTTP2.0 相对于 HTTP1.x 来说提升是巨大的，主要有以下几点：

- 二进制格式：HTTP1.x 是文本协议，而 HTTP2.0 是以二进制帧为基本单位，是一个二进制协议，一帧中除了包含数据外同时还包含该帧的标识：Stream Identifier，即标识了该帧属于哪个 request,使得网络传输变得十分灵活。
- 多路复用：一个很大的改进，原先 HTTP1.x 一个连接一个请求的情况有比较大的局限性，也引发了很多问题，如建立多个连接的消耗以及效率问题。HTTP1.x 为了解决效率问题，可能会尽量多的发起并发的请求去加载资源，然而浏览器对于同一域名下的并发

请求有限制，而优化的手段一般是将请求的资源放到不同的域名下来突破这种限制。而 HTTP2.0 支持的多路复用可以很好的解决这个问题，多个请求共用一个 TCP 连接，多个请求可以同时在这个 TCP 连接上并发，一个是解决了建立多个 TCP 连接的消耗问题，一个也解决了效率的问题。那么是什么原理支撑多个请求可以在一个 TCP 连接上并发呢？基本原理就是上面的二进制分帧，因为每一帧都有一个身份标识，所以多个请求的不同帧可以并发的无序发送出去，在服务端会根据每一帧的身份标识，将其整理到对应的 request 中。

- header 头部压缩：HTTP2.0 使用 HPACK 算法对 header 的数据进行压缩，减少请求的大小，减少流量消耗，提高效率。因为之前存在一个问题是，每次请求都要带上 header，而这个 header 中的数据通常是不变的。
- 支持服务端推送：HTTP/2.0 允许服务器未经请求，主动向客户端发送资源，这叫做服务器推送。意思是说，当我们对支持 HTTP2.0 的服务器请求数据的时候，服务器会顺便把一些客户端需要的资源一起推送到客户端，免得客户端再次创建连接发送请求到服务器端获取。这种方式非常合适加载静态资源。

服务器端推送的这些资源其实存在客户端的某处地方，客户端直接从本地加载这些资源就可以了，不用走网络，速度自然是快很多的。

## 为什么有了 HTTP 还要增加 HTTPS

HTTP 缺点：

- 通信使用明文不对数据进行加密（内容容易被窃听）
- 不验证通信方身份（容易伪装）
- 无法确定报文完整性（内容易被篡改）

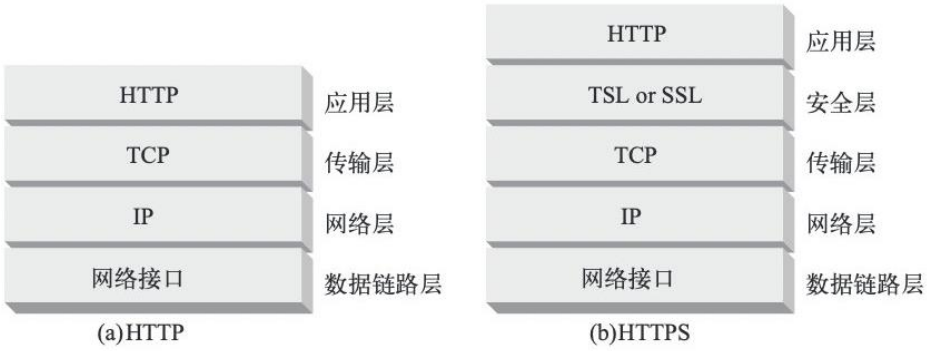
因此，HTTP 协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。(HTTP 传输密码时会使用 md5 进行加密，但 md5 很容易被破解)

为了解决 HTTP 协议的这一缺陷，需要使用另一种协议：安全套接字层超文本传输协议 HTTPS，为了数据传输的安全，HTTPS 在 HTTP 的基础上加入了 SSL（Secure Socket Layer 安全套接层）协议，SSL 依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

HTTPS 的全称是 Hypertext Transfer Protocol Secure，它用来在计算机网络上的两个端系统之间进行安全的交换信息(secure communication)，它相当于在 HTTP 的基础上加了一个 Secure 安全的词眼，那么我们可以给出一个 HTTPS 的定义：HTTPS 是一个在计算机世界里专门在两点之间安全的传输文字、图片、音频、视频等超文本数据的约定和规范。 HTTPS 是 HTTP 协议的一种扩展，它本身并不保传输的证安全性，那么谁来保证安全性呢？在 HTTPS 中，使用传输层安全性(TLS)或安全套接字层(SSL)对通信协议进行加密。也就是 HTTP + SSL(TLS) = HTTPS。

HTTPS 协议提供了三个关键的指标

- 加密(Encryption)， HTTPS 通过对数据加密来使其免受窃听者对数据的监听，这就意味着当用户在浏览网站时，没有人能够监听他和网站之间的信息交换，或者跟踪用户的活动，访问记录等，从而窃取用户信息。
- 数据一致性(Data integrity)，数据在传输的过程中不会被窃听者所修改，用户发送的数据会完整的传输到服务端，保证用户发的是什么，服务器接收的就是什么。
- 身份认证(Authentication)，是指确认对方的真实身份，也就是证明你是你（可以比作人脸识别），它可以防止中间人攻击并建立用户信任。




**HTTPS 是如何保证数据安全的？**

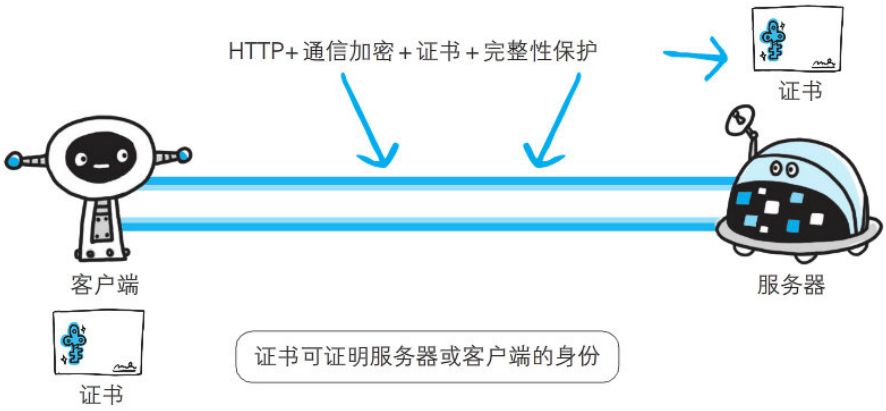
A. 客户端向服务器端发起 SSL 连接请求；（在此过程中依然存在数据被中间方盗取的可能，下面将会说明如何保证此过程的安全）

B. 服务器把公钥发送给客户端，并且服务器端保存着唯一的私钥

C. 客户端用公钥对双方通信的对称秘钥进行加密，并发送给服务器端

D. 服务器利用自己唯一的私钥对客户端发来的对称秘钥进行解密，在此过程中，中间方无法对其解密（即使是客户端也无法解密，因为只有服务器端拥有唯一的私钥），这样保证了对称秘钥在收发过程中的安全，此时，服务器端和客户端拥有了一套完全相同的对称秘钥。

E. 进行数据传输，服务器和客户端双方用公有的相同的对称秘钥对数据进行加密解密，可以保证在数据收发过程中的安全，即是第三方获得数据包，也无法对其进行加密，解密和篡改。




**图：使用 HTTPS 通信**

详细版：

A. 客户端向服务器端发起 SSL 连接请求；（在此过程中依然存在数据被中间方盗取的可能，下面将会说明如何保证此过程的安全）

B. 服务端的配置：采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面。这套证书其实就是一对公钥和私钥，如果对公钥和私钥不太理解，可以想象成一把钥匙和一个锁头，只是全世界只有你一个人有这把钥匙，你可以把锁头给别人，别人可以用这个锁把重要的东西锁起来，然后发给你，因为只有你一个人有这把钥匙，所以只有你才能看到被这把锁锁起来的东西。

C. 传送证书：这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时

间等等。

D. 客户端解析证书：这部分工作是有客户端的 TLS 来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密，就好像上面说的，把随机值用锁头锁起来，这样除非有钥匙，不然看不到被锁住的内容。（如何验证证书正确？证书是否是信任的有效证书：浏览器内置了信任的根证书，就是看看 web 服务器的证书是不是这些信任根发的或者信任根的二级证书机构颁发的。对方是不是上述证书的合法持有者：简单来说证明对方是否持有证书的对应私钥。验证方法两种，一种是对方签个名，我用证书验证签名；另外一种是用证书做个信封，看对方是否能解开。）

E. 传送加密信息：这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。

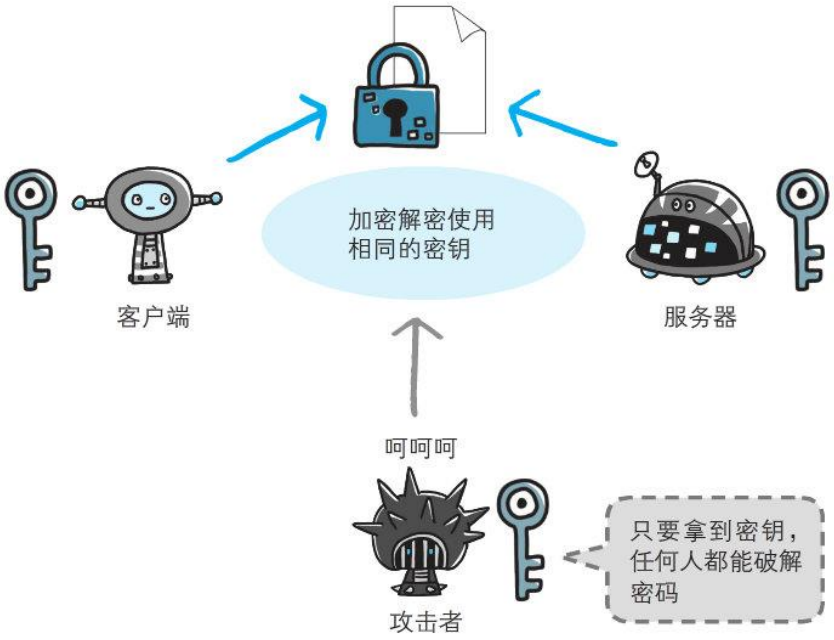
F. 服务段解密信息：服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密，所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

G. 传输加密后的信息：这部分信息是服务段用私钥加密后的信息，可以在客户端被还原。

H. 客户端解密信息：客户端用之前生成的私钥解密服务段传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听到了数据，也束手无策。

## 对称加密与非对称加密

**对称加密算法：**加密与解密使用同一个密钥，常见的对称加密算法：DES，AES，3DES 等。

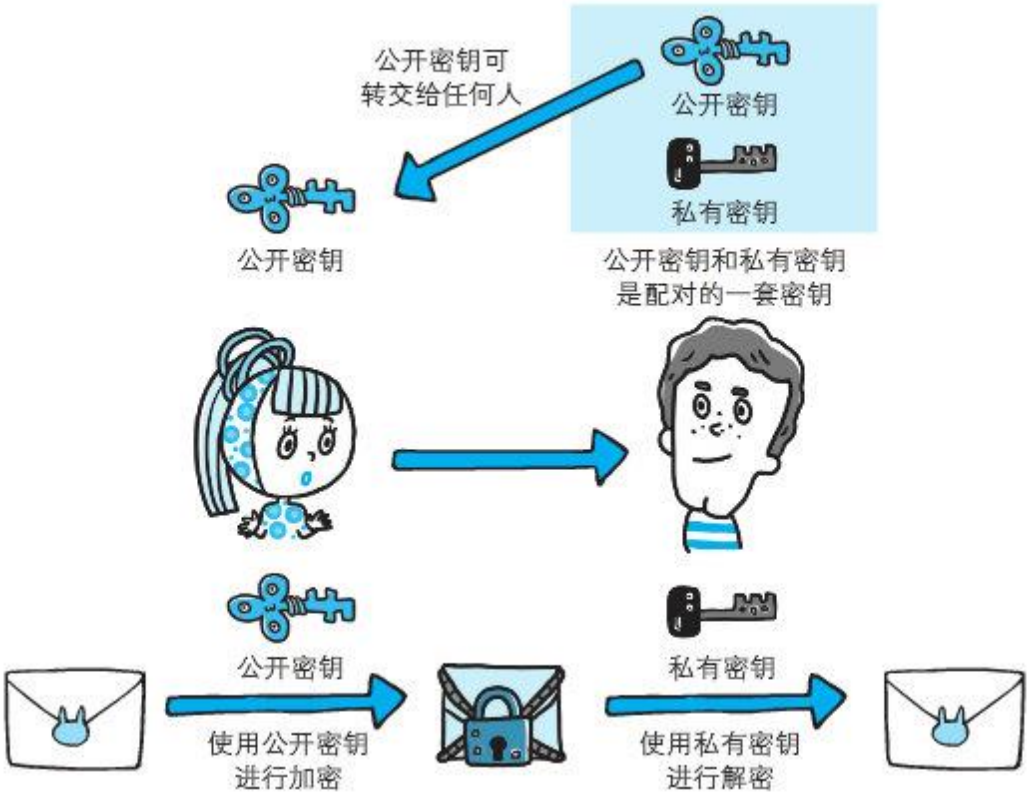



也就是说在加密的同时，也会把密钥发送给对方。在发送密钥过程中可能会造成密钥被窃取，那么如何解决这一问题呢？

**非对称加密算法：**

公开密钥使用一对非对称密钥。一把叫私有密钥，另一把叫公开密钥。私有密钥不让任何人

知道，公有密钥随意发送。公钥加密的信息，只有私钥才能解密。常见的非对称加密算法：RSA，ECC 等。
也就是说，发送密文方使用对方的公开密钥进行加密，对方接受到信息后，使用私有密钥进行解密

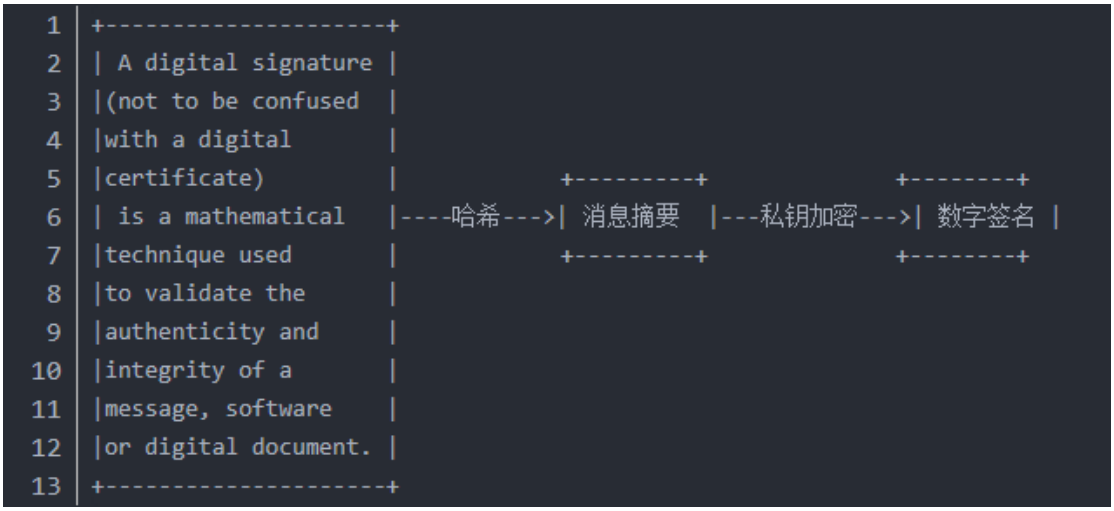



对称加密加密与解密使用的是同样的密钥，所以速度快，但由于需要将密钥在网络传输，所以安全性不高。
非对称加密使用了一对密钥，公钥与私钥，所以安全性高，但加密与解密速度慢。
为了解决这一问题，https 采用对称加密与非对称加密的混合加密方式。
SSL/TLS 协议的基本思路是采用公钥加密法，也就是说，客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密。
但是，这里有两个问题。
**如何保证公钥不被篡改？**解决方法：将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。
**公钥加密计算量太大，如何减少耗用的时间？**
每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。
因此，SSL/TLS 协议的基本过程是这样的：
服务端将非对称加密的公钥发送给客户端；
客户端拿着服务端发来的公钥，对对称加密的 key 做加密并发给服务端；
服务端拿着自己的私钥对发来的密文解密，从来获取到对称加密的 key；
二者利用对称加密的 key 对需要传输的消息做加解密传输。
**HTTPS 相比 HTTP，在请求前多了一个「握手」的环节。**

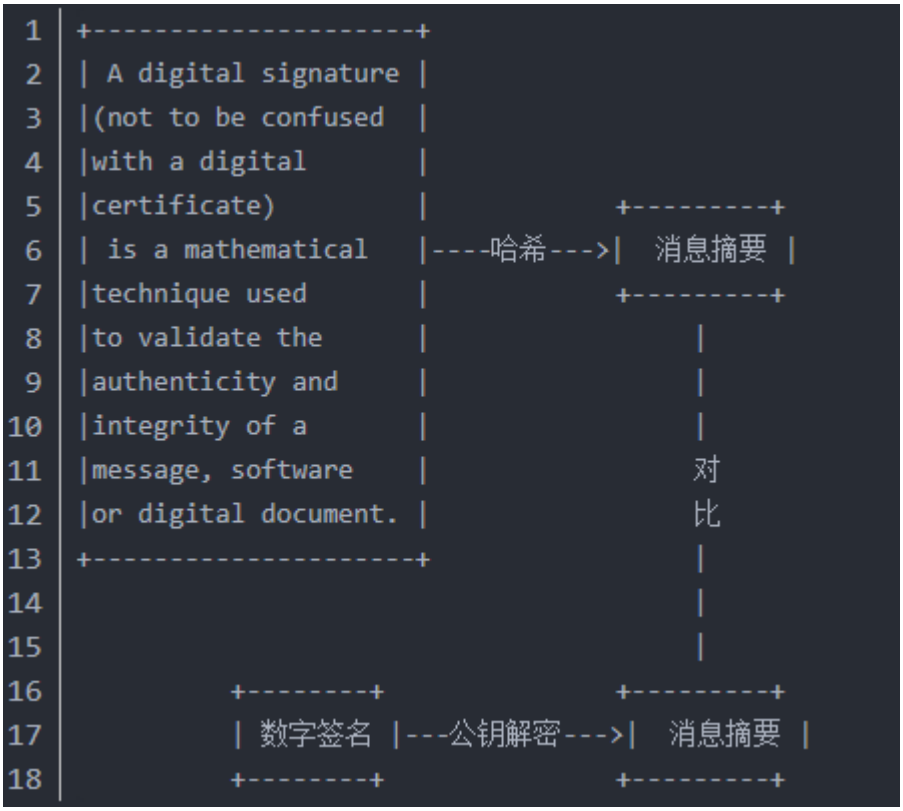
握手过程中确定了数据加密的密码。在握手过程中，网站会向浏览器发送 SSL 证书，SSL 证书和我们日常用的身份证类似，是一个支持 HTTPS 网站的身份证明，SSL 证书里面包含了网站的域名，证书有效期，证书的颁发机构以及用于加密传输密码的公钥等信息，由于公钥加密的密码只能被在申请证书时生成的私钥解密，因此浏览器在生成密码之前需要先核对当前访问的域名与证书上绑定的域名是否一致，同时还要对证书的颁发机构进行验证，如果验证失败浏览器会给出证书错误的提示。使用数字签名来验证证书是否正确

## 客户端如何验证接收到的证书

为了回答这个问题，需要引入**数字签名**(Digital Signature)。
数字签名技术是将原文通过特定 HASH 函数得到的摘要信息用发送者的私钥加密，与原文一起传送给接收者。接收者只有用发送者的公钥才能解密被加密的摘要信息，然后用 HASH 函数对收到的原文提炼出一个摘要信息，与解密得到的摘要进行对比。哪怕只是一个字符不相同，用 HASH 函数生成的摘要就一定不同。如果比对结果一致，则说明收到的信息是完整的，在传输过程中没有被修改，否则信息一定被修改过，因此数字签名能够验证信息的完整性。
一是能确定消息的不可抵赖性，因为他人假冒不了发送方的私钥签名。发送方是用自己的私钥对信息进行加密的，只有使用发送方的公钥才能解密。
二是数字签名能保障消息的完整性。一次数字签名采用一个特定的哈希函数，它对不同文件产生的数字摘要的值也是不相同的。




将一段文本通过哈希（hash）和私钥加密处理后生成数字签名。
假设消息传递在 Bob，Susan 和 Pat 三人之间发生。Susan 将消息连同数字签名一起发送给 Bob，Bob 接收到消息后，可以这样验证接收到的消息就是 Susan 发送的



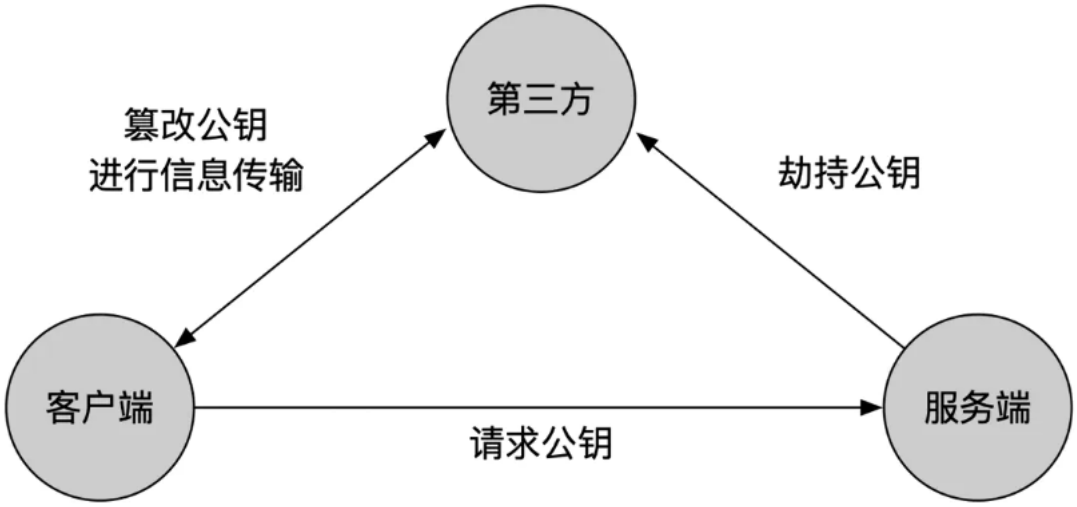

当然，这个前提是 Bob 知道 Susan 的公钥。更重要的是，和消息本身一样，公钥不能在不安全的网络中直接发送给 Bob。此时就引入了证书颁发机构（Certificate Authority，简称 CA），CA 数量并不多，Bob 客户端内置了所有受信任 CA 的证书。CA 对 Susan 的公钥（和其他信息）数字签名后生成证书。

Susan 将证书发送给 Bob 后，Bob 通过 CA 证书的公钥验证证书签名。

Bob 信任 CA，CA 信任 Susan 使得 Bob 信任 Susan，信任链（Chain Of Trust）就是这样形成的。事实上，Bob 客户端内置的是 CA 的根证书(Root Certificate)，HTTPS 协议中服务器会发送证书链（Certificate Chain）给客户端。

## 数字证书

在传输的过程中，客户端如何获得服务器端的公钥呢？当时是服务器分发给客户端，如果一开始服务端发送的公钥到客户端的过程中有可能被第三方劫持，然后第三方自己伪造一对密钥，将公钥发送给客户端，当服务器发送数据给客户端的时候，中间人将信息进行劫持，用一开始劫持的公钥进行解密后，然后使用自己的私钥将数据加密发送给客户端，而客户端收到后使用公钥解密，反过来亦是如此，整个过程中间人是透明的，但信息泄露却不得而知。

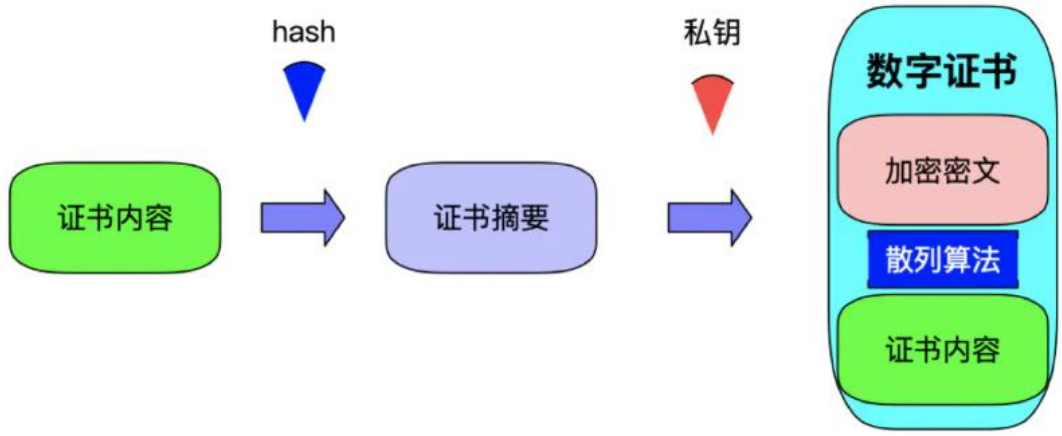



为了防止这种情况，数字证书就出现了，它其实就是基于上上面所说的私钥加密数据，公钥解密来验证其身份。

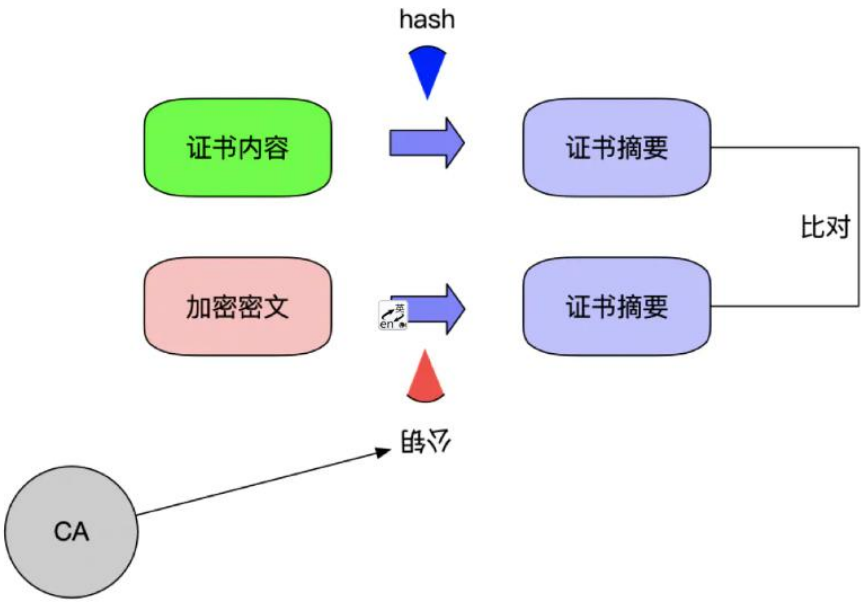
数字证书是由权威的 CA（Certificate Authority）机构给服务端进行颁发，CA 机构通过服务端提供的相关信息生成证书，证书内容包含了持有人的相关信息，服务器的公钥，签署者签名信息（数字签名）等，最重要的是公钥在数字证书中。

数字证书是如何保证公钥来自请求的服务器呢？数字证书上由持有人的相关信息，通过这点可以确定其不是一个中间人；但是证书也是可以伪造的，如何保证证书为真呢？

一个证书中含有三个部分:"证书内容，散列算法，加密密文"，证书内容会被散列算法 hash 计算出 hash 值，然后使用 CA 机构提供的私钥进行 RSA 加密。




当客户端发起请求时，服务器将该数字证书发送给客户端，客户端通过 CA 机构提供的公钥对加密密文进行解密获得散列值（数字签名），同时将证书内容使用相同的散列算法进行 Hash 得到另一个散列值，比对两个散列值，如果两者相等则说明证书没问题。




**一些常见的证书文件类型如下：**

X.509#DER 二进制格式证书，常用后缀.cer .crt

X.509#PEM 文本格式证书，常用后缀.pem

有的证书内容是只包含公钥（服务器的公钥），如.crt、.cer、.pem

有的证书既包含公钥又包含私钥（服务器的私钥），如.pfx、.p12

# DNS 协议

**参考：https://zhuanlan.zhihu.com/p/79350395**

基于 UDP 协议，DNS 的全称是 Domain Name System 或者 Domain Name Service，它主要的作用就是将人们所熟悉的网址 (域名) “翻译”成电脑可以理解的 IP 地址，这个过程叫做 DNS 域名解析。 打个比方，我们登百度的地址的时候，都是敲 www.baidu.com，进行登陆，难道你会去敲 IP 地址登百度？明显，域名容易记忆。而且，一个域名往往对应多个 DNS 地址，如下图所示

DNS 使用的是 TCP 连接还是 UDP 连接？

UDP。因为 UDP 快！UDP 的 DNS 协议只要一个请求、一个应答就好了。而使用基于 TCP 的 DNS 协议要三次握手、发送数据以及应答、四次挥手。但是 UDP 协议传输内容不能超过 512 字节。不过客户端向 DNS 服务器查询域名，一般返回的内容都不超过 512 字节，用 UDP 传输即可。

# FTP 协议

基于 TCP 协议，文件传输协议：File Transfer Protocol 早期的三个应用级协议之一，基于 C/S

结构 数据传输格式：二进制（默认）和文本 双通道协议：命令和数据连接
包含两个通道，一个是数据通道，一个是控制通道
数据通道：和 FTP 服务器进行文件传输或列表，数据连接端口为 20
控制通道：和 FTP 服务器进行沟通，连接 FTP，FTP 控制连接端口为 21
套接字 socket={IP 地址，端口号}

# SMTP 协议

SMTP 称为简单邮件传输协议（Simple Mail Transfer Protocal），目标是向用户提供高效、可靠的邮件传输。它的一个重要特点是它能够在传送中接力传送邮件，即邮件可以通过不同网络上的主机接力式传送。通常它工作在两种情况下：一是邮件从客户机传输到服务器；二是从某一个服务器传输到另一个服务器。SMTP 是一个请求/响应协议，它监听 25 号端口，用于接收用户的 Mail 请求，并与远端 Mail 服务器建立 SMTP 连接。
三个过程

A. 建立连接：在这一阶段，SMTP 客户请求与服务器的 25 端口建立一个 TCP 连接。一旦连接建立，SMTP 服务器和客户就开始相互通告自己的域名，同时确认对方的域名。
B. 邮件传送：利用命令，SMTP 客户将邮件的源地址、目的地址和邮件的具体内容传递给 SMTP 服务器，SMTP 服务器进行相应的响应并接收邮件。
C. 连接释放：SMTP 客户发出退出命令，服务器在处理命令后进行响应，随后关闭 TCP 连接。

# TELNET 协议

基于 TCP 协议

# 传输层

网络层只把分组发送到目的主机，但是真正通信的并不是主机而是主机中的进程。传输层提供了进程间的逻辑通信，传输层向高层用户屏蔽了下面网络层的核心细节，使应用程序看起来像是在两个传输层实体之间有一条端到端的逻辑通信信道。传输层主要有两个协议：TCP 与 UDP 协议

## UDP 和 TCP 的特点与区别，优缺点

用户数据报协议 UDP（User Datagram Protocol）
是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。
传输控制协议 TCP（Transmission Control Protocol）
是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。
**区别：**

- TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接。
- TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付。
- TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的，UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）。
- 每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信。
- 头部大小：TCP 首部开销 20 字节;UDP 的首部开销小，只有 8 个字节。
- TCP 的逻辑通信信道是全双工的可靠信道;UDP 则是不可靠信道。

**TCP 的优点：**可有连接的，靠，稳定 TCP 的可靠体现在 TCP 在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。
**TCP 的缺点：**慢，效率低，占用系统资源高，易被攻击 TCP 在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的 CPU、内存等硬件资源。 而且，因为 TCP 有确认机制、三次握手机制，这些也导致 TCP 容易被人利用，实现 DOS、DDOS、CC 等攻击。
**UDP 的优点：**快，比 TCP 稍安全 UDP 没有 TCP 的握手、确认、窗口、重传、拥塞控制等机制，UDP 是一个无状态的传输协议，所以它在传递数据时非常快。没有 TCP 的这些机制，UDP 较 TCP 被攻击者利用的漏洞就要少一些。但 UDP 也是无法避免攻击的，比如：UDP Flood 攻击……
**UDP 的缺点：**不可靠，不稳定 因为 UDP 没有 TCP 那些可靠的机制，在数据传递时，如果网

络质量不好，就会很容易丢包。

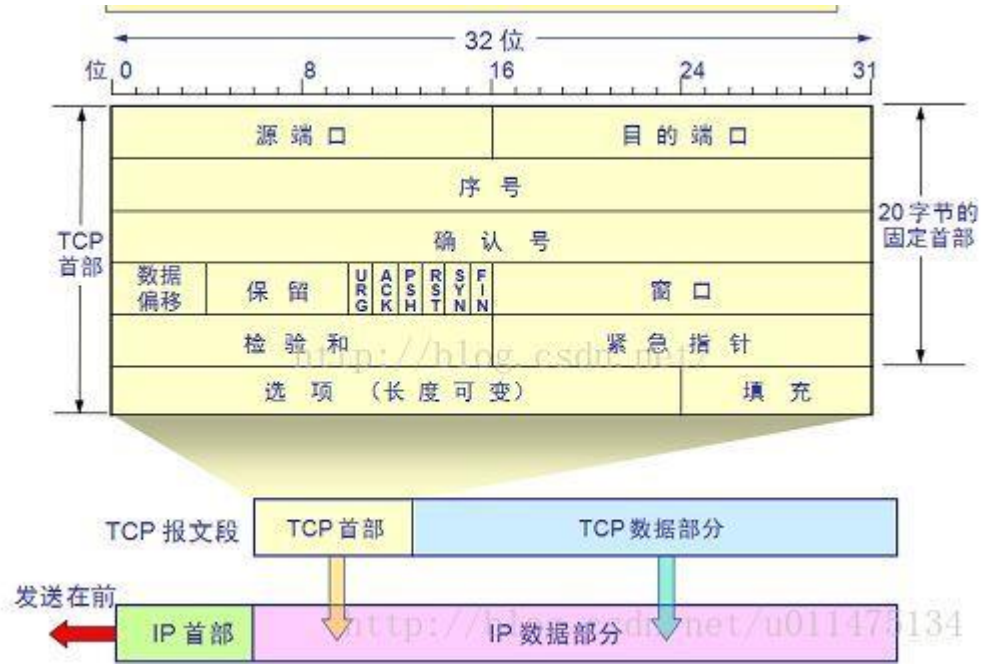
**应用场景：**

什么时候应该使用 TCP： 当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如 HTTP、HTTPS、FTP 等传输文件的协议，POP、SMTP 等邮件传输的协议。 在日常生活中，常见使用 TCP 协议的应用如下： 浏览器，用的 HTTP FlashFXP，用的 FTP Outlook，用的 POP、SMTP Putty，用的 Telnet、SSH QQ 文件传输。

什么时候应该使用 UDP： 当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快，这时就可以使用 UDP。 比如，日常生活中，常见使用 UDP 协议的应用如下： QQ 语音 QQ 视频 TFTP。

# UDP 、TCP 首部格式

UDP 首部字段只有 8 个字节，包括源端口、目的端口、长度、检验和。12 字节的伪首部是为了计算检验和临时添加的。


TCP 首部格式比 UDP 复杂。

- 序号：用于对字节流进行编号，例如序号为 301，表示第一个字节的编号为 301，如果

携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。

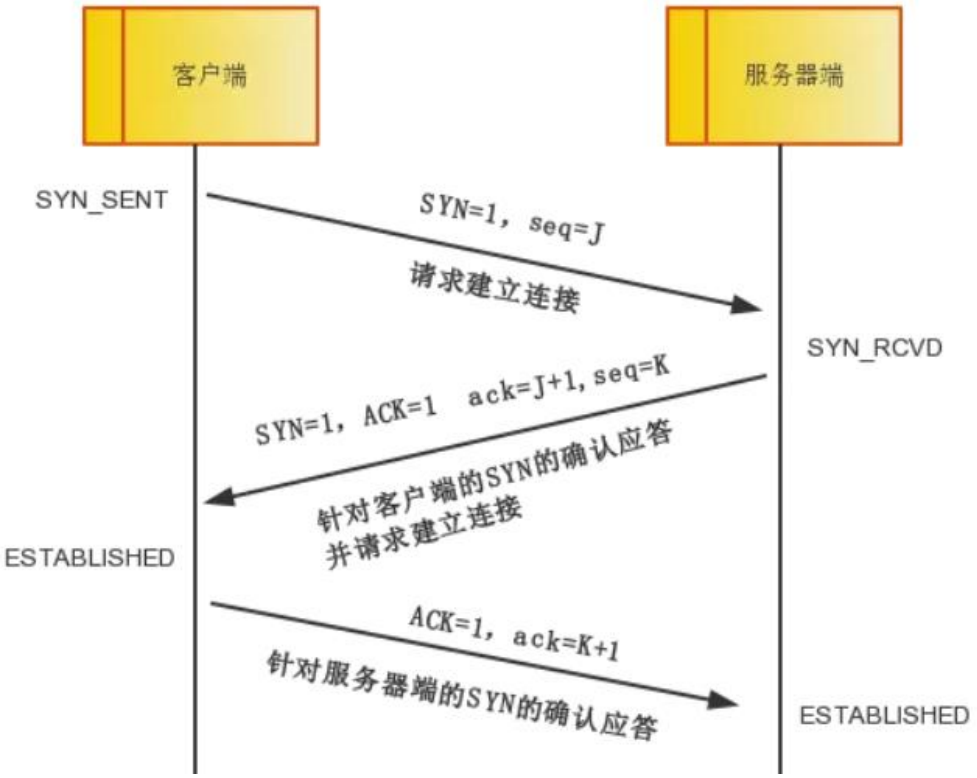
- 确认号：期望收到的下一个报文段的序号。例如 B 正确收到 A 发送来的一个报文段，序号为 501，携带的数据长度为 200 字节，因此 B 期望下一个报文段的序号为 701，B 发送给 A 的确认报文段中确认号就为 701。
- 数据偏移：指的是数据部分距离报文段头起始处的偏移量，实际上指的是首部的长度。
- 控制位：八位从左到右分别是 CWR，ECE，URG，ACK，PSH，RST，SYN，FIN。
  - CWR：CWR 标志与后面的 ECE 标志都用于 IP 首部的 ECN 字段，ECE 标志为 1 时，则通知对方已将拥塞窗口缩小；
  - ECE：若其值为 1 则会通知对方，从对方到这边的网络有阻塞。在收到数据包的 IP 首部中 ECN 为 1 时将 TCP 首部中的 ECE 设为 1；
  - URG：该位设为 1，表示包中有需要紧急处理的数据，对于需要紧急处理的数据，与后面的紧急指针有关；
  - ACK：该位设为 1，确认应答的字段有效，TCP 规定除了最初建立连接时的 SYN 包之外该位必须设为 1；
  - PSH：该位设为 1，表示需要将收到的数据立刻传给上层应用协议，若设为 0，则先将数据进行缓存；
  - RST：该位设为 1，表示 TCP 连接出现异常必须强制断开连接；
  - SYN：用于建立连接，该位设为 1，表示希望建立连接，并在其序列号的字段进行序列号初值设定；
  - FIN：该位设为 1，表示今后不再有数据发送，希望断开连接。当通信结束希望断开连接时，通信双方的主机之间就可以相互交换 FIN 位置为 1 的 TCP 段。

每个主机又对对方的 FIN 包进行确认应答之后可以断开连接。不过，主机收到 FIN 设置为 1 的 TCP 段之后不必马上回复一个 FIN 包，而是可以等到缓冲区中的所有数据都因为已成功发送而被自动删除之后再发 FIN 包；

窗口：窗口值作为接收方让发送方设置其发送窗口的依据。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

# TCP 的三次握手

所谓三次握手(Three-way Handshake)，是指建立一个 TCP 连接时，需要客户端和服务器总共发送 3 个报文。三次握手的目的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。在 socket 编程中，客户端执行 connect() 时。将触发三次握手。




A. 第一次握手：客户端将 TCP 报文标志位 SYN 置为 1，随机产生一个序号值 seq=J，保存在 TCP 首部的序列号(Sequence Number)字段里，指明客户端打算连接的服务器的端口，并将该数据包发送给服务器端，发送完毕后，客户端进入 SYN_SENT 状态，等待服务器端确认。

B. 第二次握手：服务器端收到数据包后由标志位 SYN=1 知道客户端请求建立连接，服务器端将 TCP 报文标志位 SYN 和 ACK 都置为 1，ack=J+1，随机产生一个序号值 seq=K，并将该数据包发送给客户端以确认连接请求，服务器端进入 SYN_RCVD 状态。

C. 第三次握手：客户端收到确认后，检查 ack 是否为 J+1，ACK 是否为 1，如果正确则将标志位 ACK 置为 1，ack=K+1，并将该数据包发送给服务器端，服务器端检查 ack 是否为 K+1，ACK 是否为 1，如果正确则连接建立成功，客户端和服务器端进入 ESTABLISHED 状态，完成三次握手，随后客户端与服务器端之间可以开始传输数据了。

# 为什么需要三次握手

在《计算机网络》一书中其中有提到，三次握手的目的是“为了防止已经失效的连接请求报文段突然又传到服务端，因而产生错误”，这种情况是：一端(client)A 发出去的第一个连接请求报文并没有丢失，而是因为某些未知的原因在某个网络节点上发生滞留，导致延迟到连接释放以后的某个时间才到达另一端(server)B。本来这是一个早已失效的报文段，但是 B 收到此失效的报文之后，会误认为是 A 再次发出的一个新的连接请求，于是 B 端就向 A 又发出确认报文，表示同意建立连接。如果不采用“三次握手”，那么只要 B 端发出确认报文就会认为新的连接已经建立了，但是 A 端并没有发出建立连接的请求，因此不会去向 B 端发送数据，B 端没有收到数据就会一直等待，这样 B 端就会白白浪费掉很多资源。如果采用“三次握手”的话就不会出现这种情况，B 端收到一个过时失效的报文段之后，向 A 端发出确认，此时 A 并没有要求建立连接，所以就不会向 B 端发送确认，这个时候 B 端也能够知道连接没有建立。

# 超时重传机制

- 如果第一个包，A 发送给 B 请求建立连接的报文(SYN)如果丢掉了，A 会周期性的超时重传，直到 B 发出确认(SYN+ACK)；
- 如果第二个包，B 发送给 A 的确认报文(SYN+ACK)如果丢掉了，B 会周期性的超时重传，直到 A 发出确认(ACK)；
- 如果第三个包，A 发送给 B 的确认报文(ACK)如果丢掉了，
  - A 在发送完确认报文之后，单方面会进入 ESTABLISHED 的状态，B 还是 SYN_RCVD 状态
  - 如果此时双方都没有数据需要发送，B 会周期性的超时发送(SYN+ACK)，直到收到 A 的确认报文(ACK)，此时 B 也进入 ESTABLISHED 状态，双方可以发送数据；
  - 如果 A 有数据发送，A 发送的是(ACK+DATA)，B 会在收到这个数据包的时候自动切换到 ESTABLISHED 状态，并接受数据(DATA)；
  - 如果这个时候 B 要发送数据，B 是发送不了数据的，会周期性的超时重传(SYN+ACK)直到收到 A 的确认(ACK)B 才能发送数据。

# TCP 的四次挥手（为什么四次？）

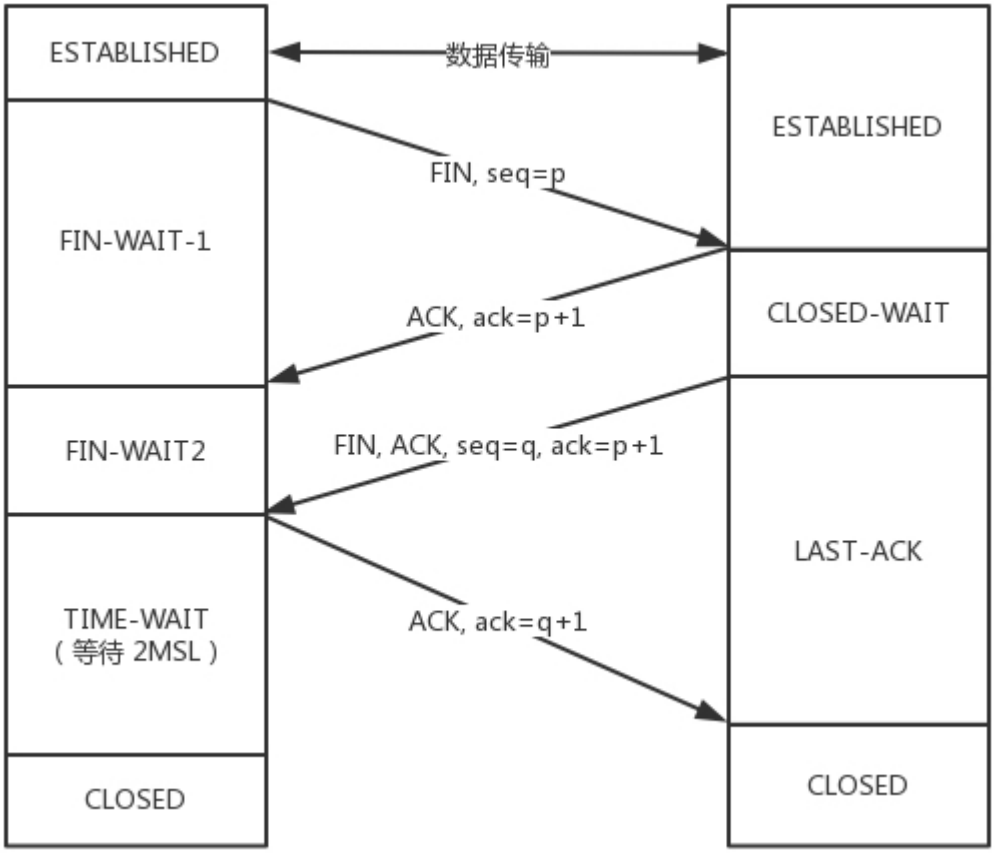
第一次挥手：A->B，A 向 B 发出释放连接请求的报文，其中 FIN（终止位） = 1，seq（序列号）=u；在 A 发送完之后，A 的 TCP 客户端进入 FIN-WAIT-1（终止等待 1）状态。此时 A 还是可以进行收数据的

第二次挥手：B->A：B 在收到 A 的连接释放请求后，随即向 A 发送确认报文。其中 ACK=1，seq=v，ack（确认号） = u +1;在 B 发送完毕后，B 的服务器端进入 CLOSE_WAIT（关闭等待）状态。此时 A 收到这个确认后就进入 FIN-WAIT-2（终止等待 2）状态，等待 B 发出连接释放的请求。此时 B 还是可以发数据的。

(如果 B 直接跑路，则 A 永远处与这个状态。TCP 协议里面并没有对这个状态的处理，但 Linux 有，可以调整 tcp_fin_timeout 这个参数，设置一个超时时间。)

第三次挥手：B->A：当 B 已经没有要发送的数据时，B 就会给 A 发送一个释放连接报文，其中 FIN=1，ACK=1，seq=w，ack=u+1，在 B 发送完之后，B 进入 LAST-ACK（最后确认）状态。

第四次挥手：A->B；当 A 收到 B 的释放连接请求时，必须对此发出确认，其中 ACK=1，seq=u+1,ack=w+1;A 在发送完毕后，进入到 TIME-WAIT （时间等待）状态。B 在收到 A 的确认之后，进入到 CLOSED（关闭）状态。在经过时间等待计时器设置的时间之后，A 才会进入 CLOSED 状态。




# 为什么需要 TIME_WAIT?

其实是客户端和服务端的两次挥手，也就是客户端和服务端分别释放连接的过程。可以看到，客户端在发送完最后一次确认之后，还要等待 2MSL 的时间。主要有两个原因，一个是为了让 B 能够按照正常步骤进入 CLOSED 状态，二是为了防止已经失效的请求连接报文出现在下次连接中。解释：

由于客户端最后一个 ACK 可能会丢失，这样 B 就无法正常进入 CLOSED 状态。于是 B 会重传请求释放的报文，而此时 A 如果已经关闭了，那就收不到 B 的重传请求，就会导致 B 不能正常释放。而如果 A 还在等待时间内，就会收到 B 的重传，然后进行应答，这样 B 就可以进入 CLOSED 状态了。在这 2MSL 等待时间里面，本次连接的所有的报文都已经从网络中消失，从而不会出现在下次连接中。

Time_wait 缺点：

正常的 TCP 客户端连接在关闭后，会进入一个 TIME_WAIT 的状态，持续的时间一般在 1~4 分钟，对于连接数不高的场景，1~4 分钟其实并不长，对系统也不会有什么影响，

但如果短时间内（例如 1s 内）进行大量的短连接，则可能出现这样一种情况：客户端所在的操作系统的 socket 端口和句柄被用尽，系统无法再发起新的连接！

举例来说：假设每秒建立了 1000 个短连接（Web 场景下是很常见的，例如每个请求都去访问 memcached），假设 TIME_WAIT 的时间是 1 分钟，则 1 分钟内需要建立 6W 个短连接，由于 TIME_WAIT 时间是 1 分钟，这些短连接 1 分钟内都处于 TIME_WAIT 状态，都不会释放，而 Linux 默认的本地端口范围配置是：net.ipv4.ip_local_port_range = 32768 61000

不到 3W，因此这种情况下新的请求由于没有本地端口就不能建立了。

消耗内存资源，CPU 资源，端口资源

Time_wait 过多解决方法：

- 开启 tw 重用，开启新的连接可以直接使用原来的 time_wait 连接
- 开启 time_wait 快速回收
- 客户端改用长连接：需要客户端的改动比较大，但能彻底解决问题，高并发的场景下，长连接的性能也明显好于短连接。
- 增加客户端的个数，避免在 2MSL 时间内使用到重复的端口。能够降低出问题概率，但需要增加成本，性价比不高。

**close_wait**

close_wait 出现在被动关闭方。出现 close_wait 只有一种情况，那就是对方发送一个 FIN 后，程序自己这边没有进一步发送 ACK 以确认。换句话说就是在对方关闭连接后，程序里没有检测到，或者程序里本身就已经忘了这个时候需要关闭连接，于是这个资源就一直被程序占用着。

这个时候快速的解决方法是：

定期扫描连接断开，检查哪些是属于 close_wait 情况

马上关闭连接

# TCP 长连接和短连接的区别

**短连接**：Client 向 Server 发送消息，Server 回应 Client，然后一次读写就完成了，这时候双方任何一个都可以发起 close 操作，不过一般都是 Client 先发起 close 操作。短连接一般只会在 Client/Server 间传递一次读写操作。

短连接的优点：管理起来比较简单，建立存在的连接都是有用的连接，不需要额外的控制手段。

**长连接**：Client 与 Server 完成一次读写之后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

如何实现？

服务器端心跳检测客户端是否存活。

在长连接的应用场景下，Client 端一般不会主动关闭它们之间的连接，Client 与 Server 之间的连接如果一直不关闭的话，随着客户端连接越来越多，Server 压力也越来越大，这时候 Server 端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可以避免一些恶意连接导致 Server 端服务受损；如果条件再允许可以以客户端为颗粒度，限制每个客户端的最大长连接数，从而避免某个客户端连累后端的服务。

# TCP 粘包、拆包及解决办法？

**为什么常说 TCP 有粘包和拆包的问题而不说 UDP ？**

由前两节可知，UDP 是基于报文发送的，UDP 首部采用了 **16bit 来指示 UDP 数据报文的长度**，因此在应用层能很好的将不同的数据报文区分开，从而避免粘包和拆包的问题。

而 TCP 是基于字节流的，虽然应用层和 TCP 传输层之间的数据交互是大小不等的数据块，但是 TCP 并没有把这些数据块区分边界，仅仅是一连串没有结构的字节流；另外从 TCP 的帧结构也可以看出，在 TCP 的首部没有表示数据长度的字段，仅有头部长度，基于上面两

点，在使用 TCP 传输数据时，才有粘包或者拆包现象发生的可能。

**什么是粘包、拆包？**

假设 Client 向 Server 连续发送了两个数据包，用 packet1 和 packet2 来表示，那么服务端收到的数据可以分为三种情况，现列举如下：

- 第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象。


- 第二种情况，接收端只收到一个数据包，但是这一个数据包中包含了发送端发送的两个数据包的信息，这种现象即为粘包。这种情况由于接收端不知道这两个数据包的界限，所以对于接收端来说很难处理。


- 第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包要么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理，对于接收端同样是不好处理的。


**为什么会发生 TCP 粘包、拆包？**

- 要发送的数据大于 TCP 发送缓冲区剩余空间大小，将会发生拆包。
- 待发送数据大于 MSS（最大报文长度），TCP 在传输前将进行拆包。
- 要发送的数据小于 TCP 发送缓冲区的大小，TCP 将多次写入缓冲区的数据一次发送出去，将会发生粘包。
- 接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包。

**粘包、拆包解决办法**

由于 TCP 本身是面向字节流的，无法理解上层的业务数据，所以在底层是无法保证数据包不被拆分和重组的，这个问题只能通过上层的应用协议栈设计来解决，根据业界的主流协议的解决方案，归纳如下：

**消息定长：**发送端将每个数据包封装为固定长度（不够的可以通过补 0 填充），这样接收端每次接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。

**设置消息边界：**服务端从网络流中按消息边界分离出消息内容。在包尾增加回车换行符进行分割，例如 FTP 协议。

**将消息分为消息头和消息体：**消息头中包含表示消息总长度（或者消息体长度）的字段。

更复杂的应用层协议比如 Netty 中实现的一些协议都对粘包、拆包做了很好的处理。

# TCP 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：
  UDP：IP 数据报大于 1500 字节,大于 MTU.这个时候发送方 IP 层就需要分片(fragmentation).把数据报分成若干片,使每一片都小于 MTU.而接收方 IP 层则需要进行数据报的重组.这样就会多做许多事情,而更严重的是,由于 UDP 的特性,当某一片数据传送中丢失时,接收方便无法重组数据报.将导致丢弃整个 UDP 数据报.
  tcp 会按 MTU 合理分片，接收方会缓存未按序到达的数据，重新排序后再交给应用层
- 流量控制：当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。滑动窗口
- 拥塞控制：当网络拥塞时，减少数据的发送。

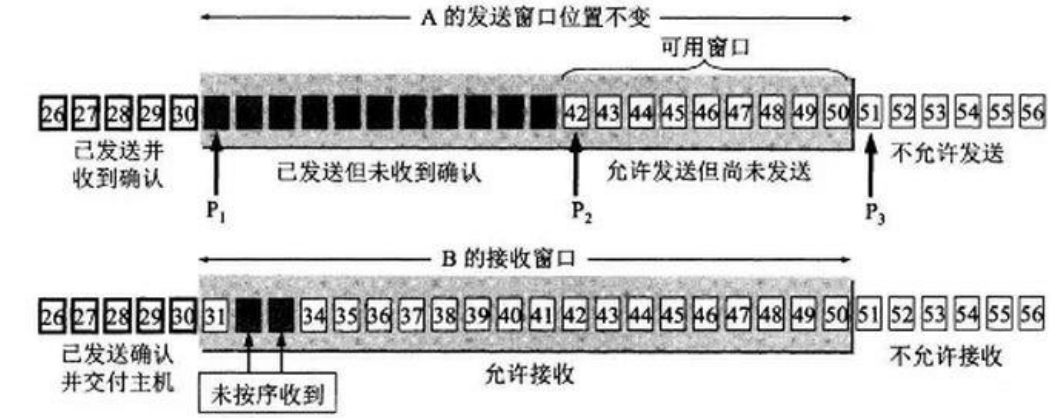
# UDP 如何保证传输可靠

UDP 不属于连接协议，具有资源消耗少，处理速度快的优点，所以通常音频，视频和普通数据在传送时，使用 UDP 较多，因为即使丢失少量的包，也不会对接受结果产生较大的影响。传输层无法保证数据的可靠传输，只能通过应用层来实现了。实现的方式可以参照 tcp 可靠性传输的方式，只是实现不在传输层，实现转移到了应用层。
最简单的方式是在应用层模仿传输层 TCP 的可靠性传输。下面不考虑拥塞处理，可靠 UDP 的简单设计。

- 添加 seq/ack 机制，确保数据发送到对端
- 添加发送和接收缓冲区，主要是用户超时重传。
- 添加超时重传机制。

# TCP 滑动窗口

窗口是缓存的一部分，用来暂时存放字节流。发送方和接收方各有一个窗口，接收方通过 TCP 报文段中的窗口字段告诉发送方自己的窗口大小，发送方根据这个值和其它信息设置自己的窗口大小。
发送窗口内的字节都允许被发送，接收窗口内的字节都允许被接收。如果发送窗口左部的字节已经发送并且收到了确认，那么就将发送窗口向右滑动一定距离，直到左部第一个字节不是已发送并且已确认的状态；接收窗口的滑动类似，接收窗口左部字节已经发送确认并交付主机，就向右滑动接收窗口。
接收窗口只会对窗口内最后一个按序到达的字节进行确认，例如接收窗口已经收到的字节为 {31, 34, 35}，其中 {31} 按序到达，而 {34, 35} 就不是，因此只对字节 31 进行确认。发送方得到一个字节的确认之后，就知道这个字节之前的所有字节都已经被接收。




# TCP 流量控制

流量控制是为了控制发送方发送速率，保证接收方来得及接收。
接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为 0，则发送方不能发送数据。
实际上，为了避免此问题的产生，发送端主机会时不时的发送一个叫做窗口探测的数据段，此数据段仅包含一个字节来获取最新的窗口大小信息。

# ARQ 协议

自动重传请求(ARQ)是 OSI 七层模型中数据链路层和传输层的一个错误纠正协议，有确认和超时两个主要机制，可以在不可靠的传输网络上实现可靠的通信。如果发送方发送消息一段时间后没有收到对应的确认帧，发送方会重新发送这条消息。

**停止等待 ARQ 协议**

发送方每发完一个分组就停止发送(暂时保留已发送分组的副本)，等待接收方的 ACK 确认帧。如果等待计时器超时后，还没收到 ACK 确认，说明消息没有成功发送，需要重新发送，直至收到对应的确认帧再发送下一个分组。
分组和分组的确认都必须进行编号，这样才能明确哪一个发送出去的分组收到了确认，而且可以检测到重复的分组，重复的分组需要丢弃，但还是需要发送确认。

这么做的优点是协议简单，缺点是信道利用率低、等待时间长。

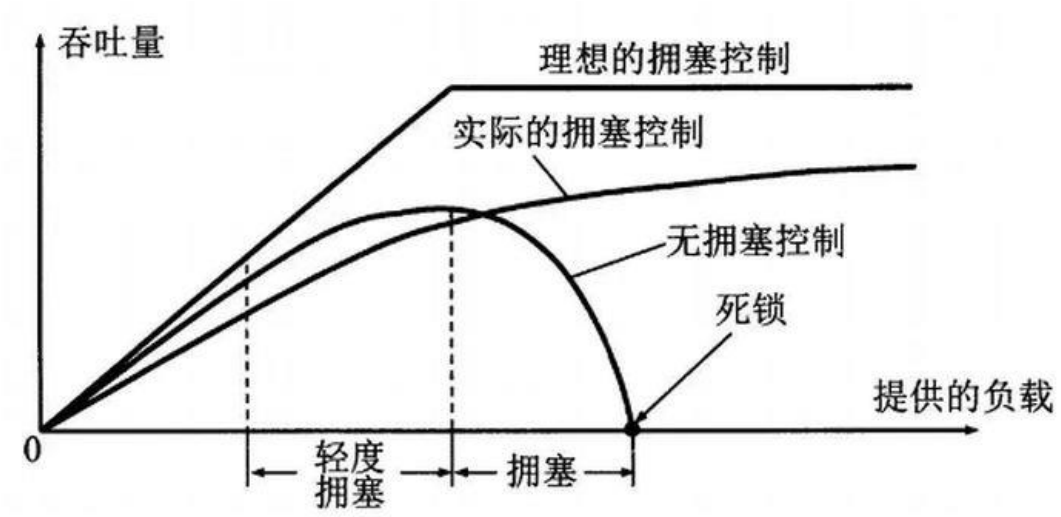
**连续 ARQ 协议**

发送方维持一个发送窗口，所有在发送窗口内的分组都可以连续发送出去，不需要停止等待接收方的确认帧。接收方采用累计确认，对按序到达的最后一个分组发送确认，表明到这个分组为止的所有分组都已经正确收到了。
这么做的优点是信道利用率高、容易实现，即使分组丢失也不必立刻重传。
缺点是不能向发送方反映出接收方已经确认收到的所有分组信息。例如：发送方发送了 5 个分组，中间的第 3 个分组丢失了，这时接收方只能对前两个分组发出确认收到，发送方无法知道后面三个分组的情况，只好把后面的三个分组都重传一次，这个过程叫做 Go-Back-

N(回退 N)，表示需要回退回来，重传已经发送过的 N 个分组。
所以当通信线路质量不好时，连续 ARQ 协议会带来负面影响。

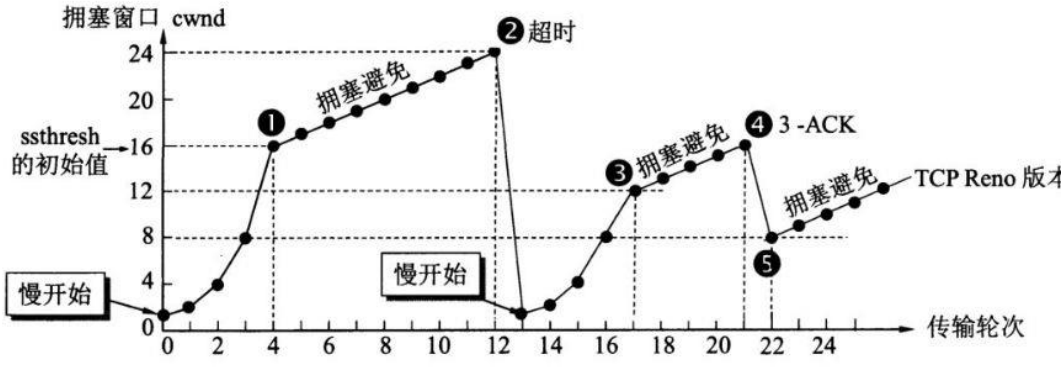
# TCP 拥塞控制

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而拥塞控制是为了降低整个网络的拥塞程度。




TCP 主要通过四个算法来进行拥塞控制：
慢开始、拥塞避免、快重传、快恢复。
发送方需要维护一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口与发送方窗口的区别：拥塞窗口只是一个状态变量，实际决定发送方能发送多少数据的是发送方窗口。
为了便于讨论，做如下假设：

- 接收方有足够大的接收缓存，因此不会发生流量控制；
- 虽然 TCP 的窗口基于字节，但是这里设窗口的大小单位为报文段。




TCP 拥塞窗口 cwnd 在拥塞控制时的变化情况

## 慢开始与拥塞避免

发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...
注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。
如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。

## 快重传与快恢复

在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。
在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。
在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。
慢开始和快恢复的快慢指的是 cwnd 的设定值，而不是 cwnd 的增长速率。慢开始 cwnd 设定为 1，而快恢复 cwnd 设定为 ssthresh。

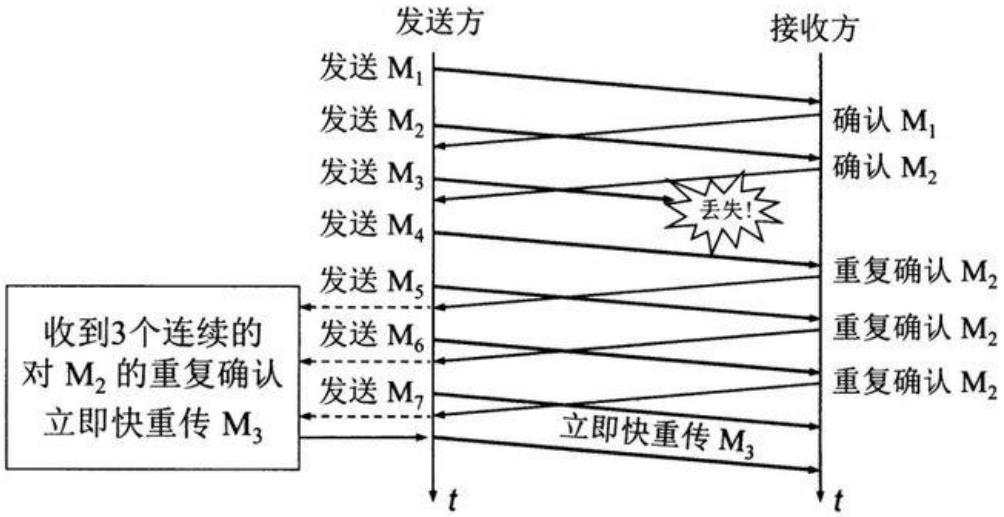


图 5-26　快重传的示意图

## TCP 端口

端口号根据范围分为三种

- Well-Known Ports（即公认端口号）

它是一些众人皆知著名的端口号，这些端口号固定分配给一些服务，我们上面提到的 HTTP 服务、 FTP 服务等都属于这一类。知名端口号的范围是：0-1023

| 应用程序 | FTP | TELNET | SMTP | DNS | TFTP | HTTP | HTTPS | SNMP |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 熟知端口号 | **21** | 23 | 25 | 53 | 69 | **80** | **443** | 161 |

- Registered Ports（即注册端口）

它是不可以动态调整的端口段，这些端口没有明确定义服务哪些特定的对象。不同的程序可以根据自己的需要自己定义，注册端口号的范围是：1024-49151

- Dynamic, private or ephemeral ports（即动态、私有或临时端口号）

顾名思义，这些端口号是不可以注册的，这一段的端口被用作一些私人的或者定制化的服务，当然也可以用来做动态端口服务，这一段的范围是：49152 – 65535

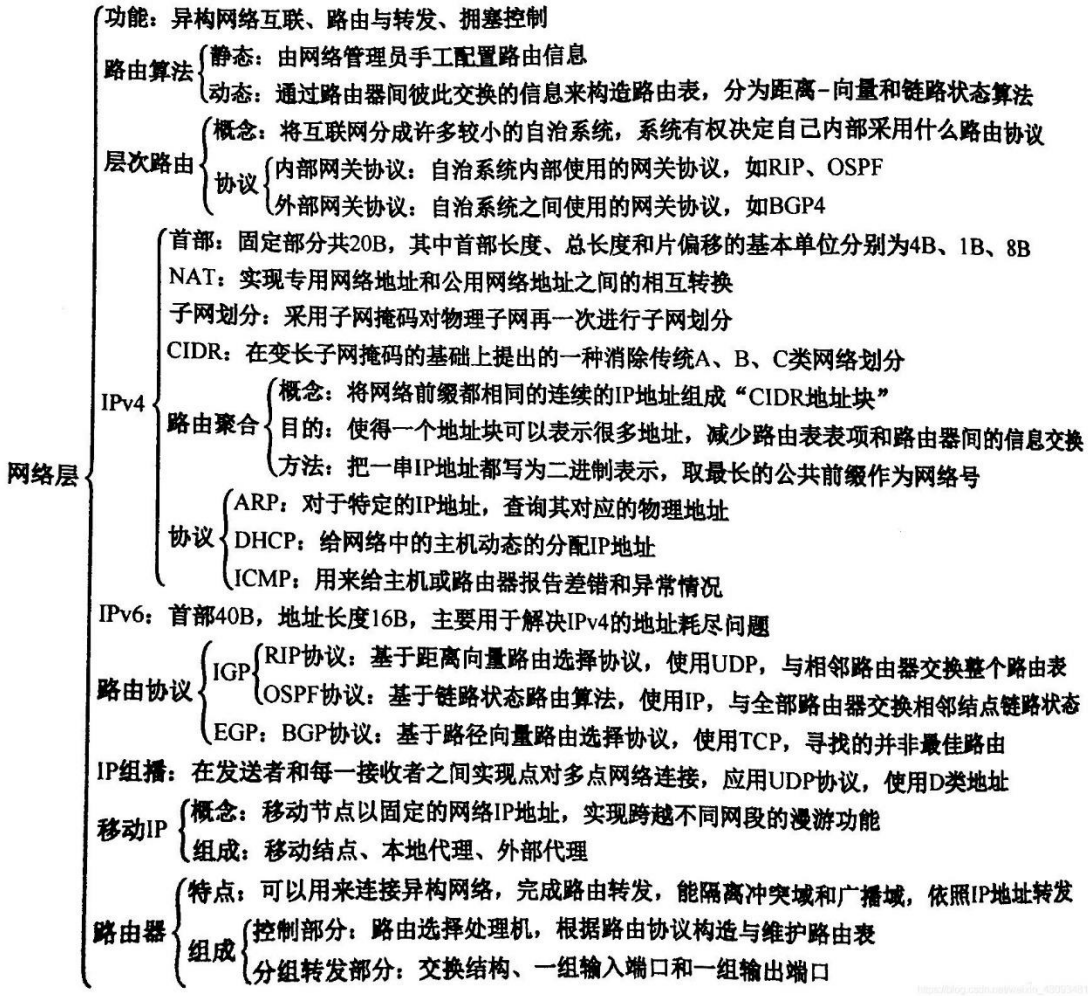
# 网络层

网络层(network layer)负责为分组交换网上的不同主机提供通信服务。 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。在 TCP/IP 体系结构中，由于网络层使用 IP 协议，因此分组也叫 IP 数据报 ，简称 数据报。

这里要注意：不要把运输层的“用户数据报 UDP ”和网络层的“ IP 数据报”弄混。另外，无论是哪一层的数据单元，都可笼统地用“分组”来表示。

网络层的另一个任务就是选择合适的路由，使源主机运输层所传下来的分株，能通过网络层中的路由器找到目的主机。这里强调指出，网络层中的“网络”二字已经不是我们通常谈到的具体网络，而是指计算机网络体系结构模型中第三层的名称.

互联网是由大量的异构（heterogeneous）网络通过路由器（router）相互连接起来的。互联网使用的网络层协议是无连接的网际协议（Intert Prococol）和许多路由选择协议，因此互联网的网络层也叫做网际层或 IP 层。



# IP 地址的划分

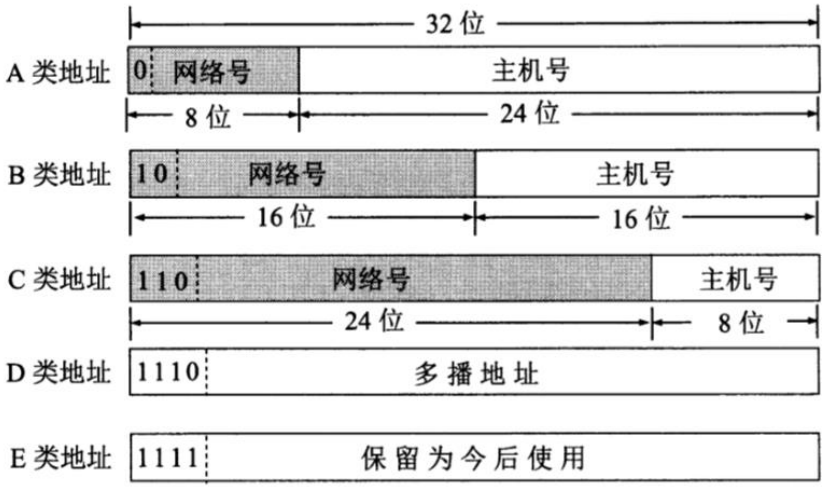



图 4-5　IP 地址中的网络号字段和主机号字段



IP 地址的指派范围：

| 网络类别 | 最大可指派的网络数 | 第一个可指派的网络号 | 最后一个可指派的网络号 | 每个网络中的最大主机数 |
|---|---|---|---|---|
| A | 126 ($2^7-2$) | 1 | 126 | 16777214 |
| B | 16383 ($2^{14}-1$) | 128.1 | 191.255 | 65534 |
| C | 2097151 ($2^{21}-1$) | 192.0.1 | 223.255.255 | 254 |

*一般不使用的特殊 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 | 在本网络上的本主机（见 6.6 节 DHCP 协议） |
| 0 | host-id | 可以 | 不可 | 在本网络上的某台主机 host-id |
| 全 1 | 全 1 | 不可 | 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 | 可以 | 对 net-id 上的所有主机进行广播 |
| 127 | 非全 0 或全 1 的任何数 | 可以 | 可以 | 用于本地软件环回测试 |



**IP 地址的重要特点：**

- 每一个 IP 地址都由网络号和主机号两部分组成，是一种分等级的地址结构
- 实际上 IP 地址是标志一个主机或(路由器)和一条链路的接口
- 用转发器或网桥连接起来的若干给局域网仍为一个网络，因为这些局域网都具有同样的网络号 net-id
- 所有分配到网络号的网络都是平等的

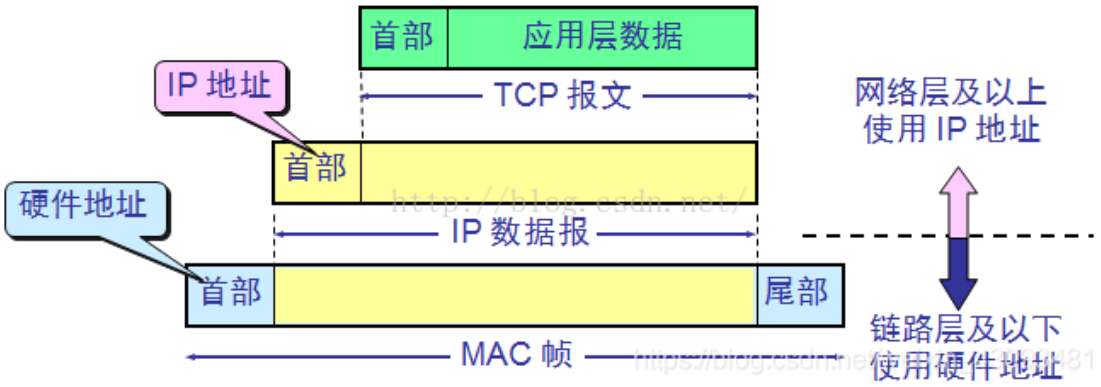
**网络上的 IP 地址：**

- 同一局域网上的各 IP 地址的网络号都是相同的
- 用网桥互连的网段仍是一个局域网，只有一个网络号
- 一个路由器有多个接口，每个接口对应的网络号不同
- 两路由器直接相连时，可以不给两端接口分配 IP，这样的特殊网络也叫无编号网络或无

名网络

**IP 地址与硬件地址**

IP 地址与硬件地址区别：从层次看，硬件地址或物理地址是数据链路层和物理层使用的地址，IP 地址是网络层和以上各层使用的地址，是一种逻辑地址




# 地址解析协议 ARP

IP 地址与 MAC 地址：源 IP 地址和目的 IP 地址始终不变；而源 MAC 地址和目的 MAC 地址在每条链路上都要变化

作用：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

工作方式：每个主机里都设有一个 ARP 高速缓存，里面有所在局域网上各主机和路由器的 IP 地址到硬件地址的映射表，且这个映射表经常动态更新

**工作流程**：当主机 A 向局域网上某个主机 B 发送 IP 数据报时，先在 ARP 高速缓存中查看有无主机 B 的 IP 地址，若有，就可查出对应的硬件地址，反之，执行 ARP 请求分组：

A. ARP 请求分组：在局域网广播一个 ARP 请求分组，包含发送方硬件地址，发送方 IP 地址，目的方硬件地址(未知时填 0)，目的方 IP 地址
B. 本地广播 ARP 请求，路由器不转发 ARP 请求
C. ARP 响应分组：某主机收到广播，发现本机 IP 与查询 IP 一致，就回复 ARP 响应分组，包含发送方硬件地址，发送方 IP 地址；同时将请求分组中 IP 与硬件地址对应关系保存，如果不一致则抛弃分组
D. 收到回复的 ARP 响应分组后，将对应 IP 和硬件地址存入 ARP 高速缓存中，方便下次使用

生存时间：ARP 高速缓存中每条映射都只存在一段时间，超过时间后就被删除

**特点：**

- ARP 协议只解决同一局域网上 IP 地址和硬件地址映射问题，不在同一局域网则无法解决
- ARP 工作过程对用户来说是透明的

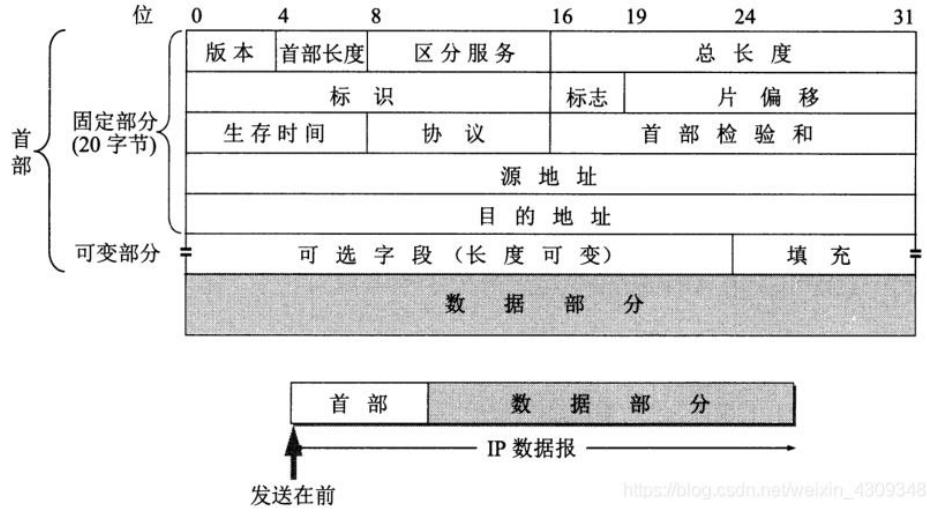
四种使用 ARP 的典型情况：

- 发送方是主机，要把 IP 数据报发送到本网络上的一个主机，这时 ARP 找到目的主机的硬件地址
- 发送方是主机，要把 IP 数据报发送到另一个网络上的一个主机，这时 ARP 找到本网络上的一个路由器的硬件地址，剩下的工作由这个路由器完成
- 发送方是路由器，要把 IP 数据报发送到本网络上的一个主机，这时 ARP 找到目的主机

的硬件地址

- 发送方是路由器，要把 IP 数据报发送到另一个网络上的主机，这时 ARP 找到本网络上的一个路由器的硬件地址，剩下的工作由这个路由器完成

# IP 数据报格式




**首部各字段含义：**

- 版本：占 4 位，说明该 IP 数据报使用的 IP 协议的版本，通信双方必须使用同一个 IP 协议版本
- 首部长度：占 4 位，长度 20~60 字节，因为 IPv4 中首部存在可变部分，所以需要指出首部的长度以划分首部与数据部分；以 4 字节为单位，不是 4 的整倍数时要填充至整倍数
- 区分服务：占 8 位，只在区分服务时才起作用，一般不用
- 总长度：占 16 位，说明该 IP 数据报的总长度(首部+数据)。IP 数据报封装为 MAC 帧时受限于 MAC 帧的长度上限，所以 IP 数据报还存在“分片”操作，即将 IP 数据报分为多片，封装进多个 MAC 帧。因此 IP 数据报最大长度 2^16-1=65535 字节
- 标识：占 16 位，IP 数据报若存在分片，则接收方需要将各分片组合出原 IP 数据报，相同标识号的 IP 数据报就说明它们其实是同一个源 IP 数据报。
- 标志：占 3 位，目前只有前两位有意义，最低位为 MF（More Fragment），若 MF=1 则说明该数据报后面“还有分片”。中间一位为 DF（Don't Fragment），若 DF=1 则不能分片，只有 DF=0 才可以分片
- 片偏移：占 13 位，用于说明该 IP 数据报（已分片）在源 IP 数据报中的相对位置（相对于数据字段的起点），单位是 8 字节，每个分片一定是 8 字节的整倍数
- 生存时间：IP 分组在网络中传递时有可能出现“兜圈子”的情况，所以需要对 IP 数据报进行一定的限制，生存时间的单位是“跳数”，最大值为 255，每经过一个路由器，路由器便将 IP 数据报的生存时间-1，当 IP 数据报中的生存时间为 0 时，路由器丢弃该分组。
- 协议：说明该 IP 数据报的上层协议类型，如 IP 对应 4，TCP 对应 6，UDP 对应 17
- 首部校验和：验证首部是否存在传输错误，只检验首部，不包括数据部分
- 源地址

- 目的地址
- 可变部分：长度为 1~40 字节，IP 地址中的可变部分可用于支持很多操作，但很多情况都用不上，而且会增加路由器处理分组的开销，所以 IPv6 中的数据报首部做成了固定长度

# IP 层转发分组的流程：

每个路由器路由表表项数：每有一个网络就要有一个路由表项。
接口所在网络为直连网络，直接交付
特定主机的路由：给特定主机指定路由表，通过指定路线访问目标主机
默认路由：可以减少路由表所占用的空间和搜索路由表所用的时间，将不在路由表中的网络都连向默认路由
路由表：只有目标网络和对应的下一跳地址，并不储存到某个网络的完整路径

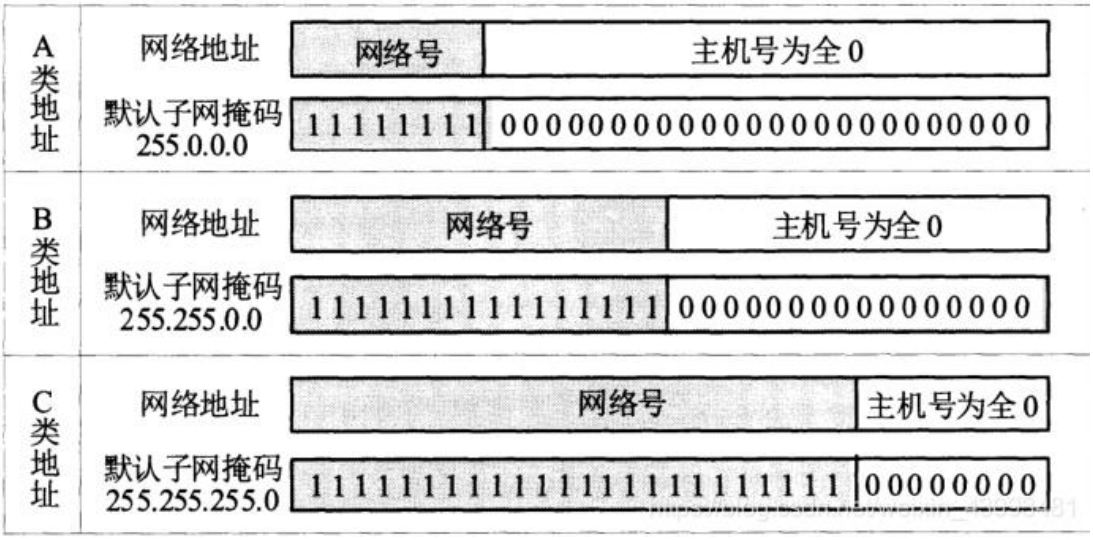
| 网络前缀 | 下一跳 |
| --- | --- |
| 131.128.56.0/24 | A |
| 131.128.55.32/28 | B |
| 131.128.55.32/30 | C |
| 131.128.0.0/16 | D |

## 不使用子网的分组转发算法：

A. 从数据报的首部提取目的主机的 **IP 地址 D**，得出目的**网络地址为 N**。
B. 若 N 就是与此路由器直接相连的某个网络地址，则进行直接交付，不需要再经过其他的路由器，直接把数据报交付给目的主机（这里包括把目的主机地址 D 转换为具体的硬件地址，把数据报封装为 MAC 帧，再发送此帧）；否则就要执行(3)进行间接交付。
C. 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中所指明的下一跳路由器，否则执行(4)。
D. 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中所指明的下一跳路由器，否则执行(5)。
E. 若路由表中有一个默认路由，则把数据报传送给路由表中所指明的下一跳路由器，否则执行(6)。
F. 报告转发分组出错。（ICMP 算法）

## 使用子网时的分组转发：

子网划分后的路由表：包含目的网络地址、子网掩码、下一跳地址




A. 从收到的数据报首部提取目的 IP 地址 D

B. 先判断是否为直接交付。对路由器直接相连的网络进行逐个检查：用各网络的子网掩码和 D 逐位相与，看结果是否和相对应的网络地址匹配。若匹配，则把分组进行直接交付，转发任务结束。否则就是间接交付，执行（3）。

C. 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中所指明的下一跳路由：否则执行（4）。

D. 对路由表的每一行，用其中的子网掩码和 D 逐位相与，其结果为 N。若 N 与该行的目的网络地址匹配，则把数据报传送给该行指明的下一跳路由器；否则执行（5）。

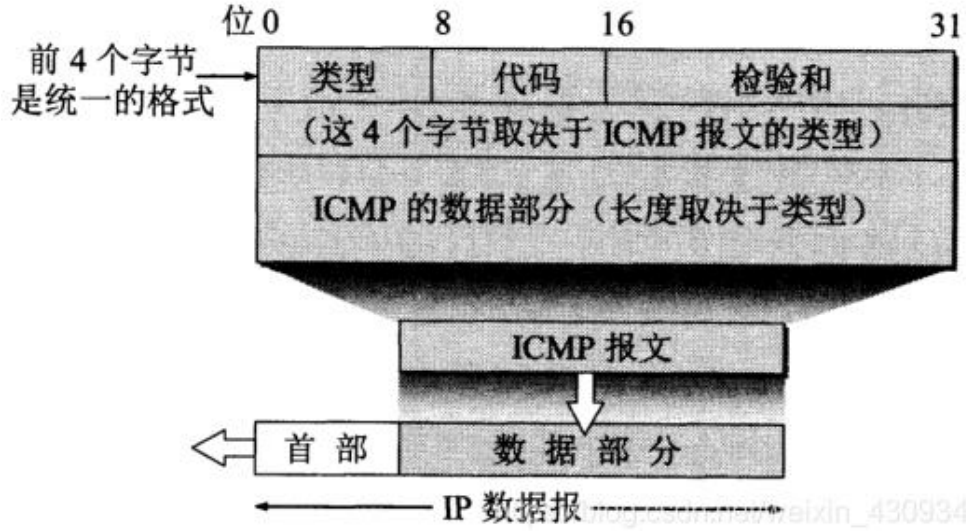
E. 若路由表中有一个默认路由，则把数据报传送给路由表中所指明的默认路由器；否则执行（6）。

F. 报告转发分组出错。

# 网际控制报文协议 ICMP

作用：ICMP 允许主机或路由器报告差错和提供有关异常情况的报告

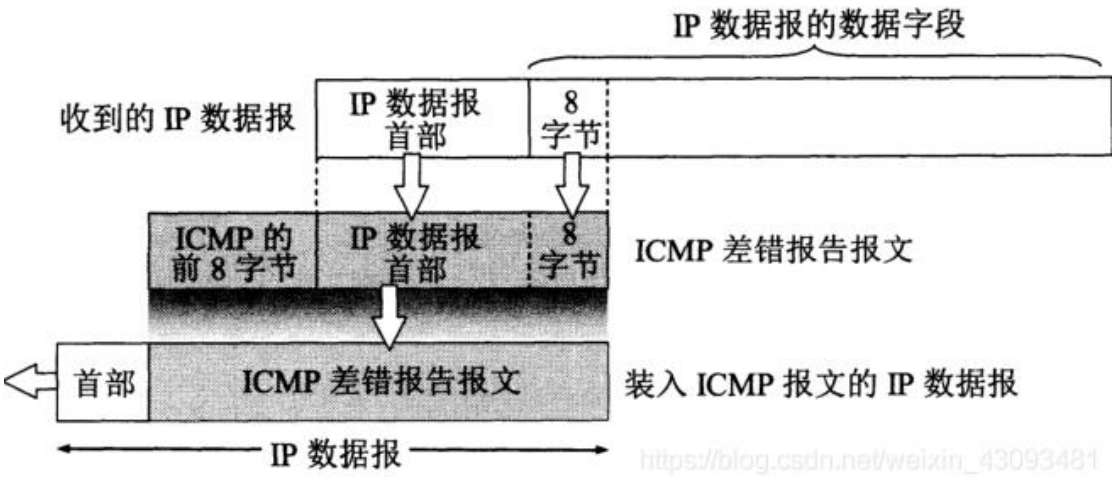
报文格式：



ICMP 报文种类：差错报告报文，询问报文

| ICMP 报文种类 | 类型的值 | ICMP 报文的类型 |
|---|---|---|
| 差错报告报文 | 3 | 终点不可达 |
| | 11 | 时间超过 |
| | 12 | 参数问题 |
| | 5 | 改变路由(Redirect) |
| 询问报文 | 8 或 0 | 回送(Echo)请求或回答 |
| | 13 或 14 | 时间戳(Timestamp)请求或回答 |

常见差错报告报文：终点不可达、时间超过、参数问题、改变路由(重定向)
ICMP 差错报告报文封装过程：




不发送 ICMP 差错报告报文的情况：

- 对 ICMP 差错报告报文，不再发送 ICMP 差错报告报文
- 对第一个分片的数据报片的所有后续数据报片，都不发送 ICMP 差错报告报文
- 对具有多播地址的数据报，都不发送 ICMP 差错报告报文
- 对具有特殊地址(如 127.0.0.0 或 0.0.0.0)的数据报，不发生 ICMP 差错报告报文
- 常用的 ICMP 询问报文：
- 回送请求和回答：ICMP 回送请求报文是由主机或路由器向一个特定的目的主机发出的询问，收到此报文的主机必须给源主机或路由器发送 ICMP 回送回答报文，这种询问报文用来测试目的站是否可达以及了解其状态
- 时间戳请求和回答：ICMP 时间戳请求报文是请某台主机或路由器回答当前的日期和时间，再 ICMP 时间戳回答报文中有一个 32 位的字段，其中写入的整数代表从 1900.1.1 到当前时刻一共多少秒，用于时钟同步和时间测量

**ICMP 的应用：**

- PING： PING 即 Packet InterNet Groper，用于探测两台主机间是否连通，源主机向目标主机发送 ICMP 的回送请求报文（封装在 IP 数据报中），目标主机若接收到该报文则返回回送回答报文
- 路由探测： 路由探测即源主机向目标主机发送无法交付的 UDP 数据报（封装于 IP 数据报，若目标主机接收到该数据报，则会返回 ICMP 终点不可达报文），第一次发送时将 IP

数据报的生存时间设为 1，这样一来第一个路由器接收到后将生存时间-1 就会直接判断该 IP 分组需要丢弃，并返回 ICMP 时间超过报文，源主机接下来发送第二个 IP 数据报（依然为不可交付 UDP 数据报），此次将生存时间设为 2……以此类推，直至接收到 ICMP 终点不可达报文，或生存时间达到上限为止。

# DHCP 协议 动态主机分配协议

DHCP 是应用层协议，基于 UDP 的，是一种让系统得以连接到网络上，并获取所需要的配置参数手段，使用 UDP 协议工作。具体用途：给内部网络或网络服务供应商自动分配 IP 地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。

DHCP 的工作原理：使用客户 / 服务器方式。需要 IP 地址的主机在启动时就向 DHCP 服务器广播发送发现报文，这时该主机就成为 DHCP 客户。本地网络上所有主机都能收到此广播报 文，但只有 DHCP 服务器才回答此广播报文。DHCP 服务器先在其数据库中查找该计算机的配置 信息。若找到，则返回找到的信息。若找不到，则从服务器的 IP 地址池中取一个地址分配给该计算机。DHCP 服务器的回答报文称为提供报文。是广播的形式。

**DHCP 服务器聚合 DHCP 客户端的交换过程如下：**

A. DHCP 客户机广播"DHCP 发现“消息，试图找到网络中的 DHCP 服务器，以便从 DHCP 服务器获得一个 IP 地址。
B. DHCP 服务器收到"DHCP 发现“消息后，向网络中广播"DHCP 提供“消息，其中包括 提供 DHCP 客户机的 IP 地址和相关配置信息。
C. DHCP 客户机收到"DHCP 提供“消息，如果接收 DHCP 服务器所提供的相关参数，那么通过广播"DHCP 请求“消息向 DHCP 服务器请求提供 IP 地址。
D. DHCP 服务器广播"DHCP 确认“消息，将 IP 地址分配给 DHCP 客户机。DHCP 允许网络上配置多台 DHCP 服务器，当 DHCP 客户机发出 DHCP 请求时，有可能收 到多个应答消息。这时， DHCP 客户机只会挑选其中的一个，通常挑选最先到达的。 DHCP 服务器分配给 DHCP 客户的 IP 地址是临时的，因此 DHCP 客户只能在一段有限的时 间内使用这个分配到的 IP 地址。DHCP 称这段时间为租用期。租用期的数值应由 DHCP 服务器 自己决定， DHCP 客户也可在自己发送的报文中提出对租用期的要求。

DHCP 的客户端和服务器端需要通过广播方式来进行交互，原因是在 DHCP 执行期间，客户端和服务器端都没有标识自己身份的 IP 地址，因此不可能通过单播的形式进行交互。采用 UDP 而不采用 TCP 的原因也很明显：TCP 需要建立连接，如果连对方的 IP 地址都不知道，那么更不 可能通过双方的套接字建立连接。

# 互联网的路由选择协议

分层次的路由选择协议：

互联网采用分层次的路由选择协议，自适应的(动态的)、分布式路由选择协议

自治系统 AS：在单一技术管理下的一组路由器，在 AS 内部使用内部网关协议，AS 之间使用外部网关协议

路由选择协议分类：

- 内部网光协议 IGP：在一个自治系统内部使用。如 RIP、OSPF 协议
- 外部网光协议 EGP：在不同自治系统之间使用。如 BGP 协议

## 内部网关协议 RIP

概述：是一种分布式，基于距离的路由选择协议
距离：直连网络距离为 1，每过一个非直连网络距离加 1，距离也称为跳数，每经过一个路由器跳数就加 1，距离实际上指最短距离
RIP 允许一个路径最多包含 15 个路由器，也就是距离最大值为 16，故 RIP 适合小型互联网使用；RIP 不能在两个网络之间同时使用多条路由
工作流程：每个路由器每隔一段时间向外广播，每个路由器收到广播后更新自己的路由表
刚开始时只知道直连网络的距离，路由表为空，以后，每个路由器只和数目有限的相邻路由器交互并更新路由信息，经过若干次更新后，所有路由器最终会知道到达本自治系统其他路由器的最短距离和下一跳地址，此时称该网络收敛
RIP 协议的特点：

- 仅和相邻路由器交换信息，不相邻的路由器不交换信息
- 交换的信息是当前本路由器所知道的全部信息，即其现在的路由表
- 按固定时间间隔交换信息

## OSPF 协议

主要特征：使用分布式的链路状态协议，而不像 RIP 使用距离向量协议
OSPF 的要点：

- 向本自治系统中所有路由器发送信息。使用洪泛法，向所有相邻路由发送信息，每个相邻路由又再将此信息发给所有相邻路由
- 发送信息就是与本路由器相邻的所有路由器的链路状态，说明与哪些路由相邻，以及该链路的"度量"；而不是发生路由表
- 只有当链路状态发生变化时，才使用洪泛法发生信息；不是定期更新
- 度量：费用、距离、时延、带宽

链路状态数据库：实际就是全网的拓扑结构图，它在全网范围内是一致的，能较快的更新，收敛较快
**OSPF 的区域：**
为使 OSPF 能够用于规模很大的网络，OSPF 将一个自治系统再划分为若干各更小的范围，叫做区域；必须要有一个主干区域，其它区域一般都和主干区域直接相连；每个区域都有一个 32 位的区域标识符；区域不能太大，一个区域路由器数量不超过 200 个；
优点：使用泛洪法交换链路信息时，仅在一个区域内而不是整个自治系统中，这减小了整个网络上的通信量
**OSPF 的特点：**

- 使用层次结构的区域划分
- OSPF 直接用 IP 数据报传送，而不用 UDP

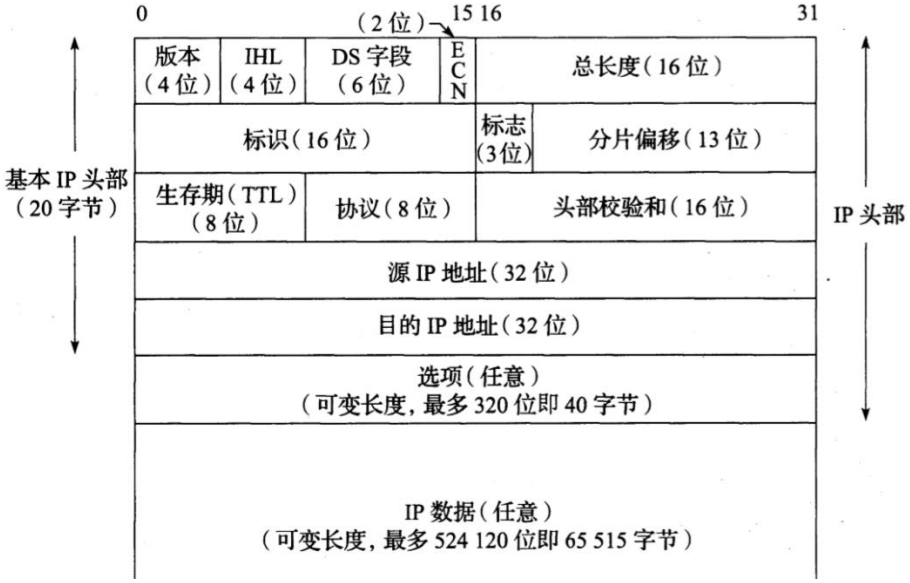
## 外部网关协议 EGP：

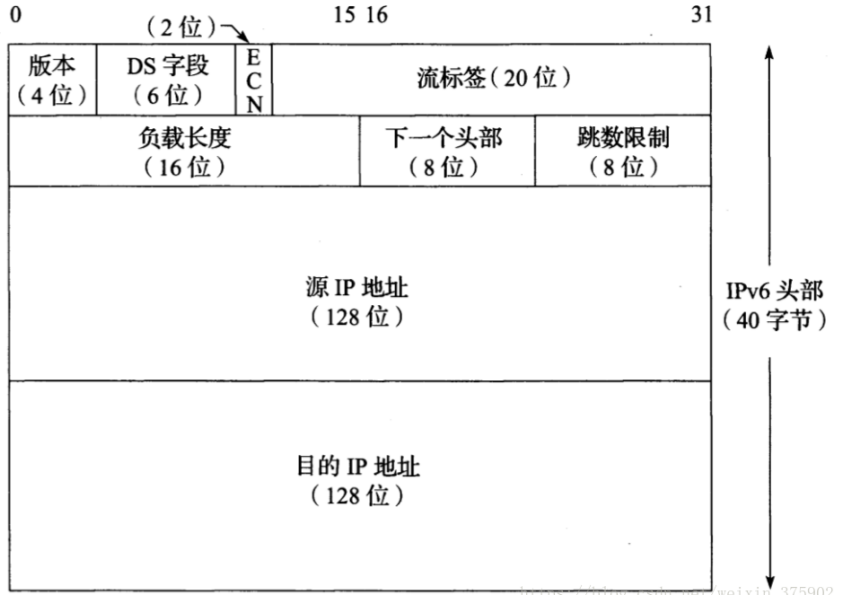
EGP 协议的作用：寻找一条能够到达目的网络且比较好的路由，不一定是最佳路由，采用路

径向量路由选择协议

# IPV4 和 IPV6 的区别

- 扩展了路由和寻址的能力
  IPv6 把 IP 地址由 32 位增加到 128 位，从而能够支持更大的地址空间
- 报头格式的简化
  IPv 4 报头格式中一些冗余的域或被丢弃或被列为扩展报头，从而降低了包处理和报头带宽的开销。虽然 IPv6 的地址是 IPv4 地址的 4 倍。但报头只有它的 2 倍大
- 身份验证和保密
  在 IPv6 中加入了关于身份验证、数据一致性和保密性的内容。

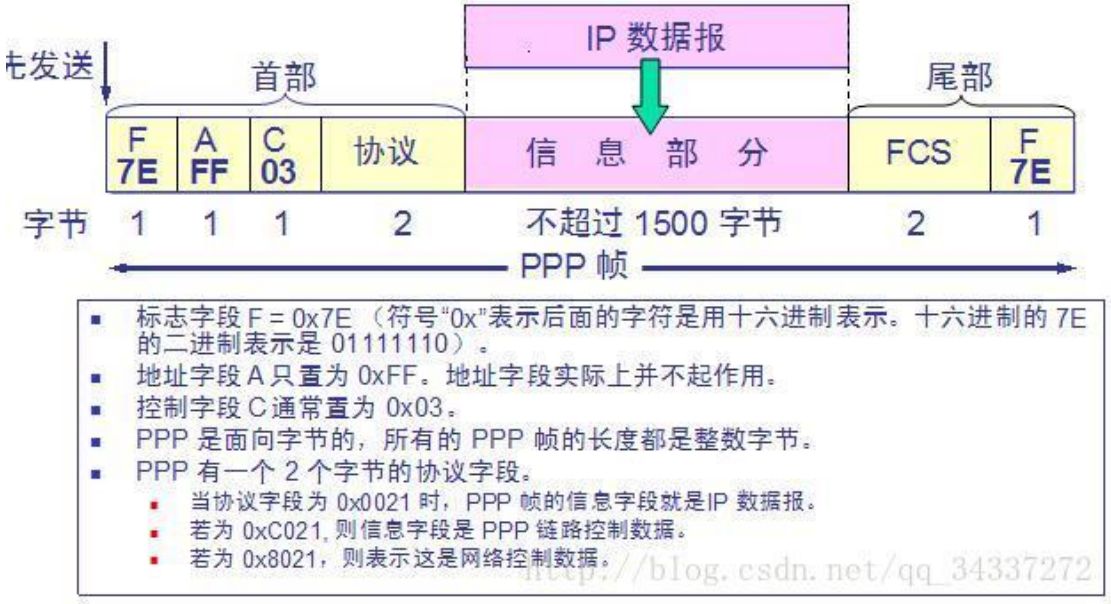


# 数据链路层

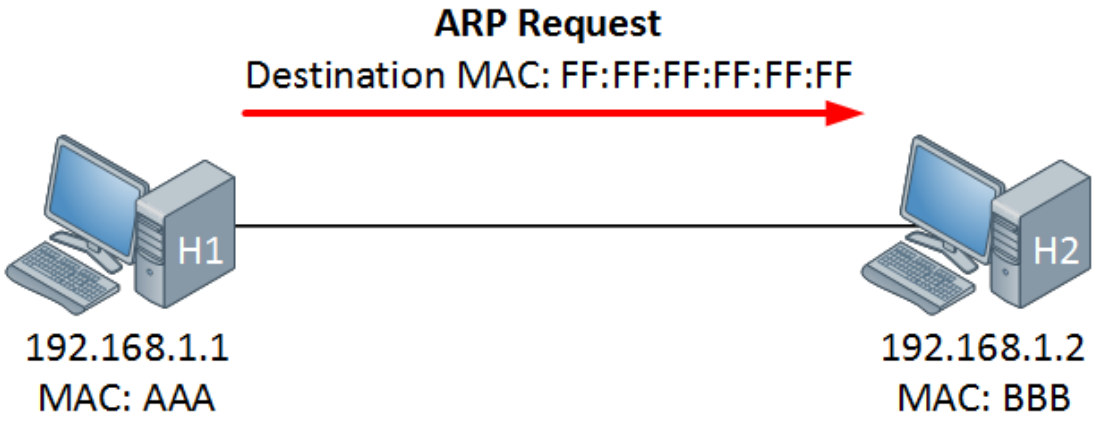
## 基本术语

链路（link）：一个结点到相邻结点的一段物理链路。
数据链路（data link）：把实现控制数据运输的协议的硬件和软件加到链路上就构成了数据链路。
循环冗余检验 CRC（Cyclic Redundancy Check）：为了保证数据传输的可靠性，CRC 是数据链路层广泛使用的一种检错技术。
帧（frame）：一个数据链路层的传输单元，由一个数据链路层首部和其携带的封包所组成协议数据单元。
MTU（Maximum Transfer Uint ）：最大传送单元。帧的数据部分的的长度上限。
误码率 BER（Bit Error Rate ）：在一段时间内，传输错误的比特占所传输比特总数的比率。
PPP（Point-to-Point Protocol ）：点对点协议。即用户计算机和 ISP 进行通信时所使用的数据链路层协议。以下是 PPP 帧的示意图：




数据链路层处于 OSI 七层模型的第二层，它定义了通过通信介质相互连接的设备之间，数据传输的规范。在数据链路层中，数据不再以 0、1 序列的形式存在，它们被分割为一个一个的“帧”，然后再进行传输。
数据链路层中有两个重要的概念：MAC 地址和分组交换。
**MAC 地址：**MAC 地址是被烧录到网卡 ROM 中的一串数字，长度为 48 比特，它在世界范围内唯一(不考虑虚拟机自定义 MAC 地址)。由于 MAC 地址的唯一性，它可以被用来区分不同的节点，一旦指定了 MAC 地址，就不可能出现不知道往哪个设备传输数据的情况。意译为媒体访问控制，或称为物理地址、硬件地址，用来定义网络设备的位置。在 OSI 模型中，第三层网络层负责 IP 地址，第二层数据链路层则负责 MAC 地址。因此一个主机会有一个 MAC 地址，而每个网络位置会有一个专属于它的 IP 地址 。地址是识别某个系统的重要标识符，“名字指出我们所要寻找的资源，地址指出资源所在的地方，路由告诉我们如

何到达该处。




**分组交换：**分组交换是指将较大的数据分割为若干个较小的数据，然后依次发送。使用分组交换的原因是不同的数据链路有各自的最大传输单元(MTU: Maximum Transmission Unit)。不同的数据链路就好比不同的运输渠道，一辆卡车(对应通信介质)的载重量为 5 吨。那么通过卡车运送 20 吨的货物就需要把这些货物分成四部分，每份重 5 吨。如果运输机的载重量是 30 吨，那么这些货物不需要分割，直接一架运输机就可以拉走。

以以太网(一种数据链路)为例，它的 MTU 是 1500 字节，也就是通过以太网传输的数据，必须分割为若干帧，每个帧的数据长度不超过 1500 字节。如果上层传来的数据超过这个长度，数据链路层需要分割后再发送。

**以太网帧：**我们用以太网举例，介绍一下以太网帧的格式。以太网帧的开头是“前导码(Preamble)”，长度为 8 字节，这一段没什么用，重点在于以太网帧的本体。

本体由首部，数据和 FCS 三部分组成：

以太网帧体格式

| 目标MAC地址 (6字节) | 源MAC地址 (6字节) | 类型 (2字节) | 数据 (46~1500字节) | FCS (4字节) |
|---|---|---|---|---|

类型部分存储了上层协议的编号，比如上层是 IP 协议，则编号为 0800。

FCS 表示帧校验序列(Frame Check Sequence)，用于判断帧是否在传输过程中有损坏(比如电子噪声干扰)。FCS 保存着发送帧除以某个多项式的余数，接收到的帧也做相同计算，如果得到的值与 FCS 相同则表示没有出错。

数据链路层使用的信道主要有以下两种类型：

- 点对点信道。这种信道使用一对一的点对点通信方式。
- 广播信道。这种信道使用一对多的广播通信方式，因此过程比较复杂。广播信道上连接的主机很多，因此必须使用专用的共享信道协议来协调这些主机的数据发

# 重要知识点总结

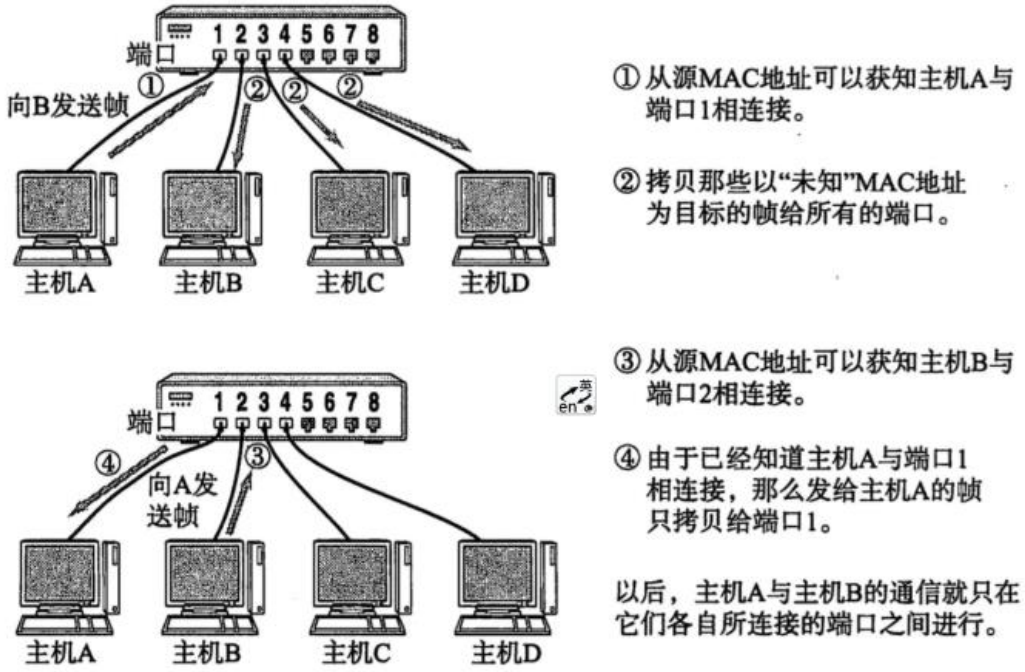
- 链路是从一个结点到相邻节点的一段物理链路，数据链路则在链路的基础上增加了一些必要的硬件（如网络适配器）和软件（如协议的实现）
- 数据链路层使用的主要是点对点信道和广播信道两种。
- 数据链路层传输的协议数据单元是帧。数据链路层的三个基本问题是：封装成帧，透明

传输和差错检测

- 循环冗余检验 CRC 是一种检错方法，而帧检验序列 FCS 是添加在数据后面的冗余码
- 点对点协议 PPP 是数据链路层使用最多的一种协议，它的特点是：简单，只检测差错而不去纠正差错，不使用序号，也不进行流量控制，可同时支持多种网络层协议
- PPPoE 是为宽带上网的主机使用的链路层协议
- 局域网的优点是：具有广播功能，从一个站点可方便地访问全网；便于系统的扩展和逐渐演变；提高了系统的可靠性，可用性和生存性。
- 计算机与外接局域网通信需要通过通信适配器（或网络适配器），它又称为网络接口卡或网卡。计算器的硬件地址就在适配器的 ROM 中。
- 以太网采用的无连接的工作方式，对发送的数据帧不进行编号，也不要求对方发回确认。目的站收到有差错帧就把它丢掉，其他什么也不做
- 以太网采用的协议是具有冲突检测的载波监听多点接入 CSMA/CD。协议的特点是：发送前先监听，边发送边监听，一旦发现总线上出现了碰撞，就立即停止发送。然后按照退避算法等待一段随机时间后再次发送。 因此，每一个站点在自己发送数据之后的一小段时间内，存在这遭遇碰撞的可能性。以太网上的各站点平等的争用以太网信道
- 以太网的适配器具有过滤功能，它只接收单播帧，广播帧和多播帧。
- 使用集线器可以在物理层扩展以太网（扩展后的以太网仍然是一个网络）
- 数据链路层的点对点信道和广播信道的特点，以及这两种信道所使用的协议（PPP 协议以及 CSMA/CD 协议）的特点
- 数据链路层的三个基本问题：封装成帧，透明传输，差错检
- 以太网的 MAC 层硬件地址
- 适配器，转发器，集线器，网桥，以太网交换机的作用以及适用场合

# 交换机

交换机是一种在数据链路层工作的网络设备，它有多个端口，可以连接不同的设备。交换机根据每个帧中的目标 MAC 地址决定向哪个端口发送数据，此时它需要参考“转发表”
转发表并非手动设置，而是交换机自动学习得到的。当某个设备向交换机发送帧时，交换机将帧的源 MAC 地址和接口对应起来，作为一条记录添加到转发表中。
下图描述了交换机自学过程的原理




关于数据链路层，最重要的一点还是它的定义：“通过通信介质相互连接的设备之间，数据传输的规范”。这说明数据链路层的协议适用于处于同一种数据链路两端的节点。如果不能理解这一点，就无法理解网络层和 IP 协议。

数据链路层的意义在于，如果没有数据链路层，数据只能以流的形式存在与通信介质中，不知道该发送往哪里，过长的数据流可能无法在通信介质中传输。

# 物理层：

基本概念物理层解决如何在连接各种计算机的传输媒体上传输数据比特流，而不是指具体的传输媒体，尽可能的屏蔽掉传输媒体和通信手段的差异。

物理层的主要任务描述为：确定传输媒体的接口的一些特性

- 机械特性：接口形状，引线数目
- 电气特性：规定电压的范围（-5V 到 +5V）
- 功能特性：例：规定 -5V 表示 0，+5V 表示 1
- 过程特性：各个相关部件的工作步骤

**信道：**信道一般表示向一个方向上传递信息的媒体，我们平时说的通信线路往往包含一条发送信息的信道和一个接受信息的信道

- 单工通道：只有一个方向上的通信，而没有反方向的交互
- 半双工通信：不能同时发送，只能一个发送，一个接受（例如：对讲机）
- 全双工通信：通信双方可以同时发送和接受（例如：手机）