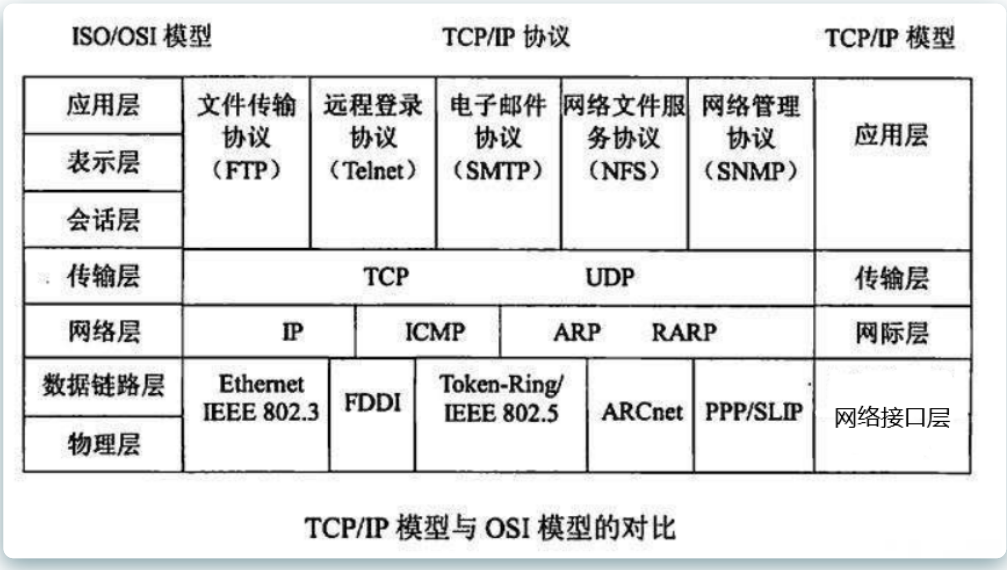
计算机网络篇

✓ 网络模型



TCP/IP 模型与 OSI 模型的对比

OSI模型每层的作用及相关协议/硬件：

对应层	作用	传输形式	协议/硬件
应用层	提供用户服务	报文	http、https、ftp、SMTP
表示层	对数据的进行**格式转换**、**加密**	报文	
会话层	管理通信会话	报文	
传输层	管理点到点的可靠数据传输	数据段	TCP、UDP
网络层	进行**路由选择**，决定传输路径	数据报	IP、ICMP、ARP、RARP
数据链路层	管理相邻两个设备之间的通信	帧	
物理层	提供数据传输的**介质**	比特流	中继器、集线器

✓ MAC地址和IP地址

MAC地址：6B(48bit)，十六进制表示，是网络上每个设备不同**接口**的唯一标识，在**数据链路层封装**。

IP地址：4B(32bit)，点分十进制表示，是网络跟主机的唯一标识，在**网络层封装**。

> 数据包转发过程中，源IP地址和目的IP地址不变，而源MAC地址和目的MAC地址逐跳改变。

✓ ARP地址解析协议

ARP：可以通过IP地址获取到对应设备的MAC地址。（连接IP跟MAC的桥梁）

RARP：可以通过MAC地址获取到对应的IP地址。

> 每台主机都有个ARP高速缓存表，记录了IP地址跟MAC地址的对应关系。

在主机之间要发送数据包时，会根据在下一跳的IP地址，在本机的ARP缓存表查找下一跳的MAC地址，如果没找到，会通过广播形式获取到下一跳的MAC地址并记录在缓存表中。

✓ ICMP网际控制报文协议

主机可通过ICMP发送 差错报文 和 询问报文。

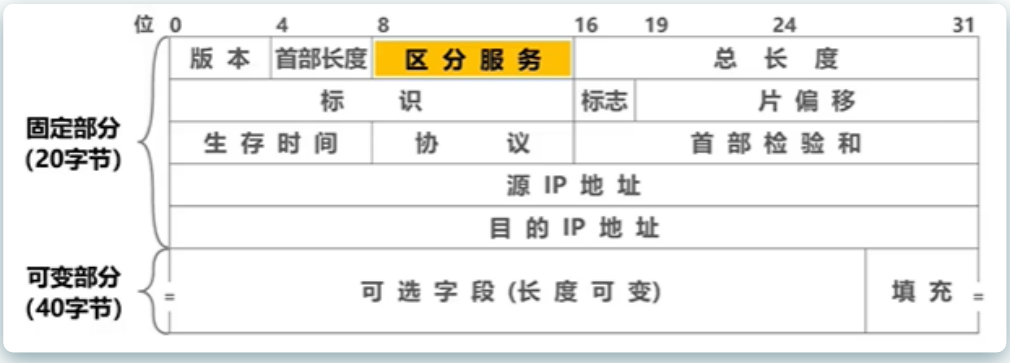
> ping命令是利用**询问报文**来探测网络之间的连通性。

> traceroute命令是利用**差错报文**和**TTL生存时间**来测试IP数据报从源主机到达目的主机要经过哪些路由器。数据报每到达一个路由器，TTL就减一，TTL=0时路由会向源主机发送差错报文。TTL一开始设置为1，然后每次递增。

✓ IP数据报首部格式

每个IP数据报由 数据报头部+数据区 组成

IP**数据报头部**格式如下（固定部分为20B）：



版本号：指定IP协议版本，IPv4还是IPv6

首部长度：IP数据包头部的长度，以4B为单位，即 该值*4=头部真实长度

服务类型：声明了数据报在网络系统传输时可以被怎样处理。

总长度：整个IP数据报的长度，最长65535

> 每个帧的数据载荷的长度是有限的，如果某个数据报的总长度太大时，会对其进行分片，每片单独作为一个数据报。而标识、标记、片位移三个字段共同协助完成切片的动作。
>
> 如数据载荷的最大长度是500，数据长度为1000；能否给每个数据报分成500？不可以，因为还需要给每个数据报加上头部数据20B。

标识：每个数据报的标识。属于同个数据报的 分片数据报 应该具有相同的标识。

标记：表示是否允许分片或者是否是最后一个分片。DF=0表示允许分片，MF=0表示最后一个分片。

片位移：分片数据报 在 原数据报 的偏移量。

生存时间：指定数据报在网络中传输的最长时间。每到达一个路由器，TTL就减一。可以防止数据报在网络中一直转发。

协议标识：指明该数据报使用的是什么协议。

协议名称	ICMP	TCP	UDP
字段值	1	6	17

首部校验和：对首部的有效性进行校验。每经过一个路由器就要重新计算首部校验和（TTL会变）。

> 校验和原理：
>
> 发送端首先将检验和字段**置0**，然后对头部中**每16位**二进制数进行**累加求和后取反**，把计算结果作为校验和。如果累加后超出16位，则把超出的数加到低位。
>
> 接收方进行验证的时候，也是对头部中每16位二进制数进行累加求和后取反，**结果为0表示验证成功**。

源IP地址：32位，发送端IP地址

目的IP地址：32位，目的端IP地址

填充字段：补充0，保证首部长度是4的倍数

✓ TCP报文段首部格式

每个TCP报文段由 **首部+数据载荷** 组成。

TCP报文段首部格式如下：

源端口：发送方的端口号

目的端口：接收方的端口号

序号：本TCP报文段的数据载荷的第一个字节的序号。

确认号：ACK=1时该字段才有效。表示前n-1个序号的数据都已收到，期望下个报文段的第一个数据的序号为n。

数据偏移：实际是报文段的首部长度，以4B为单位，即 该值*4=头部真实长度

保留：留给以后使用

TCP标记位：

- URG：紧急位，使紧急指针有效
- ACK：确认位，使确认号生效
- PSH：推送位，尽快把数据交给应用层
- RST：复位标记位，重新建立连接
- SYN：同步标记位，为1表示它是个连接请求报文
- FIN：结束标记位，为1表示它是个释放连接报文

窗口：指定接收方的发送窗口大小，用该字段来进行流量控制。

校验和：同IP数据报的校验和

紧急指针：指定紧急数据在数据载荷中的位置。

填充：保证首部长度是4的倍数

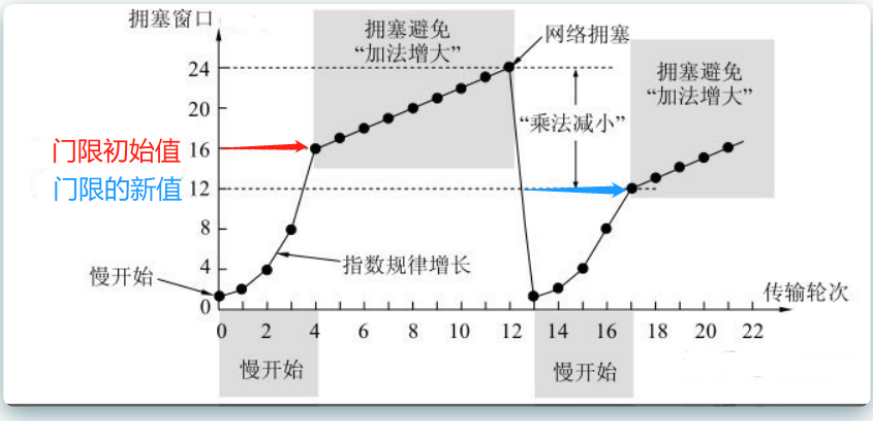
✓ TCP和UDP的区别

1. TCP是面向连接的，提供可靠的服务，而UDP是无连接的，不保证可靠交付。
2. 每一条TCP连接只能是一对一通信；UDP支持多种通信方式。

3. TCP面向字节流，传输的时候以字节为单位；UDP是面向报文的，直接发送整个数据包。
4. TCP的首部长度比UDP的大。TCP首部最小是20B，UDP固定是8B。

✓ TCP如何保证可靠

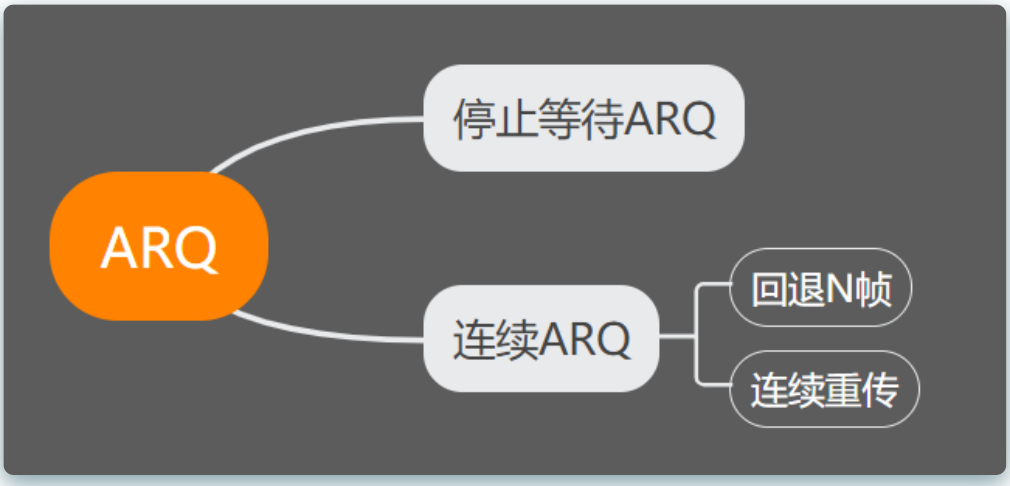
- 确认应答与序列号
  - 每次接收方收到数据后，都会对传输方进行确认应答，在应答报文中会带着序列号。
- 自动超时重传
  - 发送方发出一个报文后，会启动一个**重传计时器**，如果在规定时间内没有收到对方的应答，会重发这个报文。
- 三次握手和四次挥手
- 流量控制
  - 因为双方都有个缓冲空间，超过这个空间的报文将会丢失，所以要实时控制滑动窗口的大小。通过改变**滑动窗口**的大小来改变发送的数据量。
  - 当滑动窗口变为0时，会启动**持续计时器**，防止改变滑动窗口的报文丢失。如果持续计时器超时，则会发送窗口探测报文，获取窗口大小。
- 拥塞控制
  - 通过慢启动算法跟拥塞避免算法来进行拥塞控制。一开始使用**慢启动算法**，如果网络不拥堵就将拥塞窗口以指数级(2^n)增长；当达到慢启动门限值的时候，改为使用**拥塞避免算法**，只要网络不拥堵，则试探性的增大拥塞窗口的大小（每次加一）。
  - 无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞，就要把**慢启动门限值 设置为出现拥塞时的发送方窗口大小的一半**（但不能小于2），然后重新进入慢开始阶段。



> 怎么判断网络是否发生拥堵？**没有出现数据包发生超时重传**的情况就说明网络不拥堵。
>
> 门限值指的是什么？包的数量还是包的数据量？**包的数量**。
>
> 拥塞控制是控制整个网络中包的数量；流量控制是控制通信中包的数据量，避免数据量超过缓冲区的大小。

✓ ARQ协议

ARQ，**自动重传请求**（Automatic Repeat-reQuest）是OSI模型中**传输层**的协议之一。它通过 确认应答 和 超时 这两个机制，在不可靠服务的基础上实现可靠的信息传输。ARQ包括**停止等待ARQ协议**和**连续ARQ协议**，连续ARQ又分为**回退N帧ARQ协议**、**选择性重传ARQ协议**。



停止等待ARQ协议

停止等待协议就是**每发完一个分组就停止发送，等待对方确认，直到收到确认后再发下一个分组**。如果超过了重传时间还是没有收到 ACK 确认，说明没有发送成功，需要重新发送。

在该协议下，发送方每发出一个报文后，都会启动一个**重传计时器**，如果在规定时间内没有收到对方的应答，会重发这个报文。

缺点：效率太低。

> 重传时间应比数据传输的平均往返时间RTT更长一些，因为发送跟接收过程要花费一点时间。

如果ACK确认报文**丢失**了，发送方也会进行重传，此时接收方收到重复分组，会丢弃该分组，然后发送ACK确认包。

如果ACK确认报文**超时**了，发送方也会进行重传，此时接收方收到重复分组，会丢弃该分组，然后发送ACK确认包，发送方收到确认后继续发送下一个报文。后续发送方收到之前超时的ACK报文时会丢弃该报文。

回退N帧ARQ协议

> 为了克服停止等待ARQ协议长时间等待ACK的缺点，连续ARQ协议会连续发送一组数据包，然后再等待这些数据包的ACK。
>
> 连续ARQ一次性能发的数据包的数量主要取决于**滑动窗口**的大小和**拥塞窗口**大小。

在该协议下，发送方每发出一组报文后，都会启动一个**重传计时器**，如果在规定时间内没有收到对方的应答，会重发这个报文。

1. 接收端会**丢弃**从第一个没有收到的数据包往后的所有数据包（即使收到也丢弃）
2. 当在一定时间内没有收到某个数据包的ACK时，发送端会重新发送那个没有ACK的数据包开始往后的数据包
3. 接收方每次发送ACK，都是告诉发送方前面的n个数据包都收到了。

选择重传ARQ协议

1. 发送端连续发送数据包，且为每个数据包都设有个一个计时器
2. 当在一定时间内没有收到某个数据包的ACK时，发送端只重新发送那个没有ACK的数据包

✓ TCP字节序是大端还是小端字节序

字节序类型：整数在内存中保存的顺序。不同CPU有不同的字节序类型。

最常见的两种字节序类型：

- **大端**（Big endian）：数据是从右到左表示的。例如二进制的表示：0011
- **小端**（Little endian）：数据是从左到右表示的。例如十进制的表示：1100

所有的网络字节序都是大端字节序。

✓ TCP粘包

TCP 粘包是指：**发送方发送的若干个 数据报 被接收方接收时，被当成一个数据报来处理**。

TCP 是基于字节流的，多个数据包存储于连续的缓存中，在对数据包进行读取时由于**无法确定发送方的发送边界，**而采用某一估测值大小来进行数据读出，若双方的size不一致时就会使数据包的边界发生错位，导致读出错误的数据分包，进而曲解原始数据含义。

粘包出现原因

1. 发送方粘包：发送方需要等到缓冲区满了才会发送，若连续几次发送的数据都很少，通常TCP会根据优化算法把这些数据合成一包后一次发送出去，这时出现粘包现象。

2. 接收方粘包：接收方不及时的接收，导致数据堆在一起

什么时候不需要考虑粘包

1. 如果每次tcp双方发送完一段数据后，就关闭连接，这样就不会出现粘包问题。
2. 如果发送数据无结构。如文件传输，这样发送方只管发送，接收方只管接收存储，也不用考虑粘包

如何避免粘包?

避免粘包有以下两种方式：

- 在每个包的末尾加上特殊字符，用以区分连续的两个包。（例如加 `\r\n` 标记）
- 在报文首部添加包的长度。（可选字段中可以添加）

✓ 一个 TCP 连接可以对应几个 HTTP 请求

1. 如果tcp连接保持长连接 Connection:keep-alive；则只要在tcp连接（默认两小时）不断开，可以一直串行发送请求，数量没有上限；
2. 如果tcp连接不保持长连接，Connection:close 只能发一次请求；

✓ 一个 TCP 连接中 HTTP 请求可以一起发送么（比如一起发三个请求，再三个响应一起接收）

在 HTTP/1.1 存在 Pipelining 技术可以完成这个多个请求同时发送，但是由于浏览器默认关闭，所以可以认为这是不可行的。

在 HTTP2 中由于 Multiplexing 特点的存在，多个 HTTP 请求可以在同一个 TCP 连接中并行进行。

✓ HTTP2简单介绍

相对于Http1.x的文本格式传输， `HTTP/2` 引入 `二进制数据帧` ，数据帧对数据进行顺序标识，这样接收方收到数据之后，就可以按照序列对数据进行合并。同样是因为有了序列，服务器就可以并行的传输数据 即多路复用。

HTTP2 中，**同域名下所有通信都在单个连接上**（即同一个TCP管道）完成，该连接可以承载任意数量的双向数据流，当出现**丢包现象时，后面会阻塞**，即后面的数据将接收不了。

> HTTP3可以进行更快速的连接以及解决队头阻塞问题（多条管道）。

✓ 为什么有的时候刷新页面不需要重新建立 SSL 连接?

因为TCP 连接有时候是维持一段时间的，即不需要重新建立连接，SSL 自然也会用之前的。

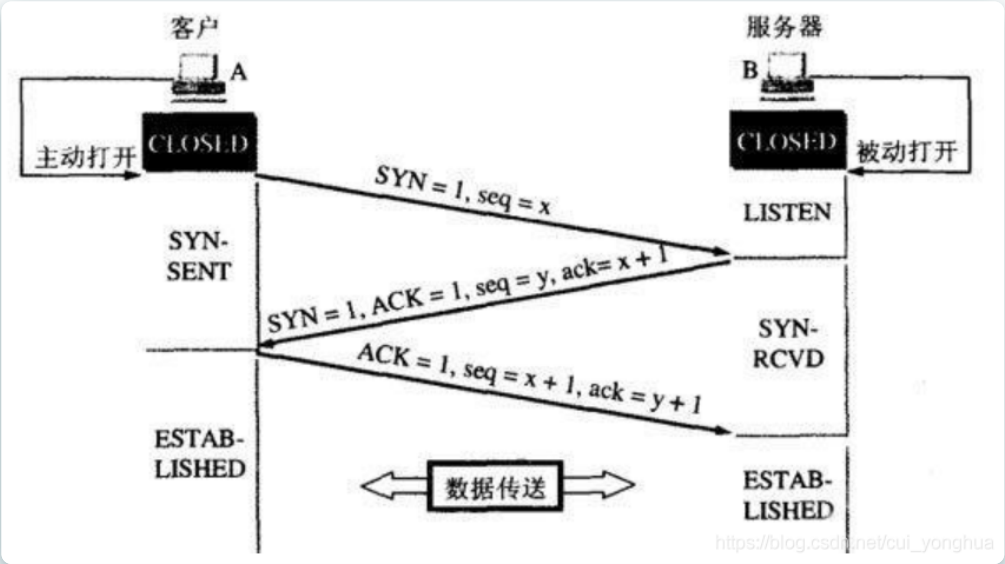
✓ 简单描述一下，TCP的连接和释放过程

三次握手的过程

1. 主机A向主机B发送TCP连接请求报文（其中报文中同步标志位SYN=1，序号seq=x），主机A进入同步等待状态。
2. 主机B收到请求后，回复连接确认报文。（其中确认报文中，SYN=1，ACK=1，并含主机B的初始序列号seq(B)=y，以及确认号ack(B)=x+1），主

机B进入同步已接收状态。

3. 主机A收到主机B的确认报文后，还需最后发送一个报文，即发送一个ACK=1，序列号seq(A)=x+1；确认号为ack(A)=y+1的报文；完成连接



> **带有SYN标志的过程包是不可以携带数据的**,也就是说三次握手的前两次是不可以携带数据的。
>
> 所以只有第三次握手才可以携带数据。因为第一次携带数据的话服务器容易被攻击。
>
> 第一次握手的seq是随时间变化的，为了防止在网络中被延迟的连接报文又发送到服务器，导致服务器解析错误。

为什么需要三次握手?

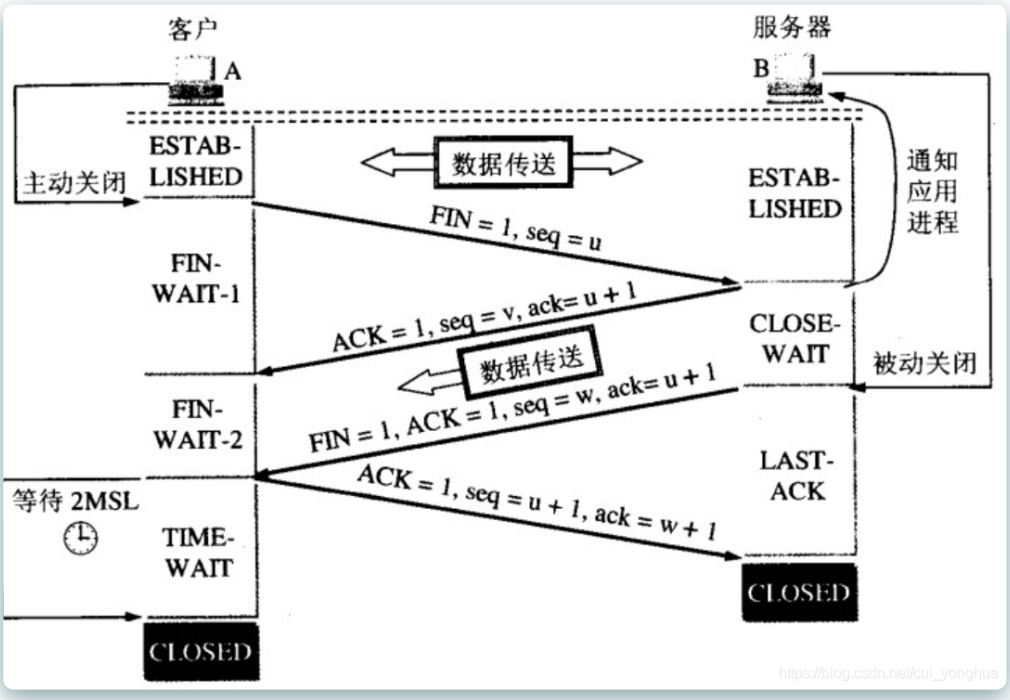
TCP的三次握手最主要是 **防止已过期的报文再次传到服务器后导致连接错误**。

> 三次握手实质上是建立TCP连接的同时确定双方的序列号和确认号以及确保双方可以正常通信。

四次挥手过程

假设主机A为客户端，主机B为服务器，其释放TCP连接的过程如下：

1. 首先客户端发送一个请求断开报文，用来**关闭**客户端到服务器的**数据传送**，然后等待服务器的确认。其中 终止标志位FIN=1，序列号seq=u。
2. 服务器收到数据包之后发回一个确认报文，ACK=1，确认号ack=u+1。此时服务器还可以向客服端发送数据。
3. 当服务器发送完数据之后，会再次发送一个请求断开报文给客户端，表示**关闭**服务器到客户端的**连接**（FIN=1，ACK=1，seq=v，ack=u+1）
4. 客户端收到报文后，发回一个确认报文（ACK=1，seq=u+1，ack=v+1）。此时客户端接着进入等待状态并启用**等待计时器**，等待2MSL后进入关闭状态。



> 为什么要四次挥手?
>
> 保证通信双方的数据都已发送完毕。因为双方发送的速率不一样，只有等待双方数据都发送完之后才能关闭连接。先发送完的一方先发送FIN包，对方确认应答后可以继续向 先发送完的一方 发送数据。

为什么要先进入TIME-WAIT状态，等待2MSL时间后才进入CLOSED状态?

为了保证服务器能收到客户端的确认应答。若A发完确认应答后直接进入CLOSED状态，那么如果该应答报文丢失，服务器等待超时后就会重新发送连接释放请求，但此时客户端已经关闭了，不会作出任何响应，此时B永远无法正常关闭。

为什么要等待2MSL

MSL表示数据包的最大存活时间。

若A发送的确认报文在一个MSL内还没有发到B，则B会**重新发送一个数据包**。这样一来一回的最大时间就是2MSL。

并且等待2MSL之后，可以**确保所有的数据包都已经失效**，不会对重新连接的新报文跟老报文发生冲突。

✓ 对称加密与非对称加密

对称加密是指加密和解密使用同一个密钥，这种方式存在的最大问题就是密钥如何安全地将密钥发给对方；

- DES、AES

非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的**公钥进行加密**，对方接收到加密信息后，使用自己的**私钥进行解密**。

- RSA

✓ Http工作原理

HTTP协议（HyperText Transfer Protocol，超文本传输协议）在进行网络通信时双方要遵循的规则。

> 超文本：不仅仅是文本，还可以是图片、视频等

HTTP请求响应模型

HTTP的请求和响应模型：**浏览器/服务器模型（B/S）**

> C/S：客户端/服务器模型，类似于APP，客户端既要做逻辑处理也要做页面展示。服务器发生一次升级，所有客户端的程序都需要升级。
>
> B/S：浏览器/服务器模型，页面渲染展示交给浏览器，逻辑处理交给服务器。

HTTP是一个无状态的协议。无状态是指 当浏览器向服务器发送请求，服务器返回响应之后，连接就被关闭了，服务器不保留与连接有关的信息。

但可以通过设置 `Connection: keep-alive` 来保持一段时间的连接。从HTTP/1.1起，**默认都开启了Keep-Alive**，简单地说，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。

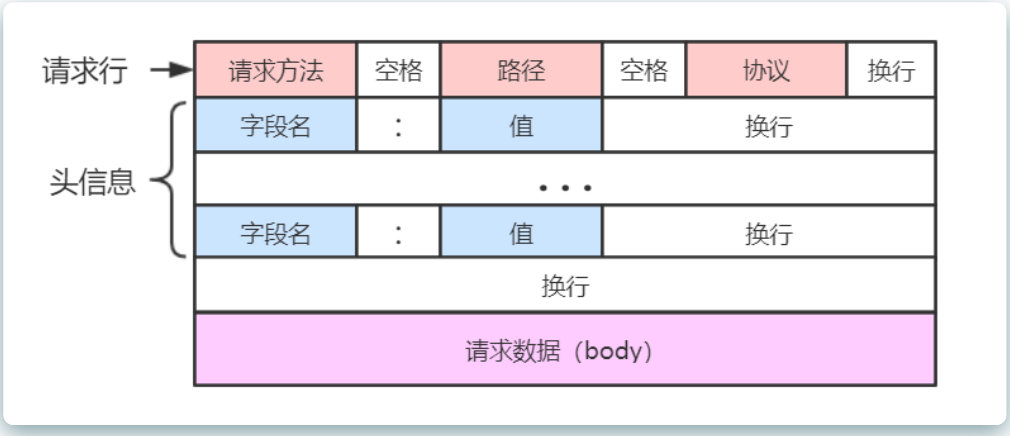
HTTP工作过程

地址栏输入域名网址后回车：

1. 地址解析：
    1.1 查看**浏览器缓存**有无该域名的IP
    1.2 查看**本机的DNS缓存**，看有没有对应域名的IP地址（位于 `C:\Windows\System32\drivers\etc\hosts` ）
    1.3 到**本地域名服务器**查询得到IP地址
2. 将请求解析成HTTP标准的报文格式
3. 三次握手，建立TCP连接
4. 发送请求，得到数据，在浏览器中渲染
5. 四次挥手，关闭TCP连接

Http1.x请求报文格式

Http1.x请求报文格式固定如下：



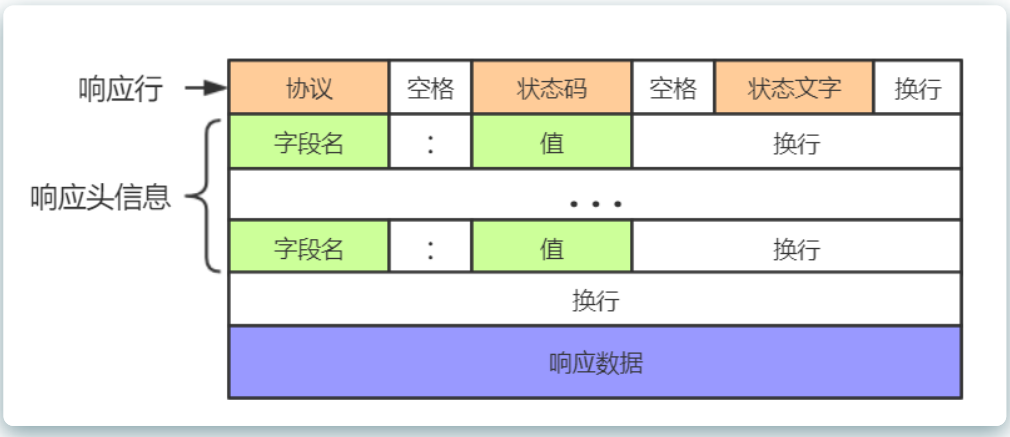
例如：（**post请求时必须指定 Content-length**）

```
POST /abc/a.html HTTP/1.1
content-type: application/json; charset=utf-8
Content-length:20

username=admin&age=3
```

Http响应报文格式

Http响应报文格式固定如下：



例如：

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Content-Type: text/html;charset=utf-8

```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Document</title>
</head>
<body>
    <p>哈哈哈哈</p>
</body>
</html>
```

Http请求方法

方法	作用
GET	请求页面信息，获得数据
HEAD	类似于get请求，只不过获取的是HTTP的头信息
POST	POST请求可能会导致新的资源的建立或已有资源的修改
PUT	更新服务器资源
DELETE	删除服务器资源
OPTIONS	获取服务器可以请求的方法

GET和POST的区别

1. **参数传递方式**：GET参数通过URL传递，POST放在Request body中。因为很多浏览器的URL长度是有限制的，所以get参数的大小也有限制。
2. **请求编码**：GET请求只能进行url编码 `application/x-www-form-urlencoded` ，而POST支持多种编码方式，如 url编码、 `application/json` 、 `multipart/form-data` 。
3. **浏览器回退**：GET在浏览器回退时单纯回到页面，而POST会再次提交请求。
4. **浏览器缓存**：GET请求会放入浏览器cache中，而POST不会，除非手动设置。

✓ 响应码

1. 1开头：表示成功接收请求，要求客户端**继续提交**下一次请求才能完成整个处理过程。比如发送post请求前，需要先发送一个请求，询问服务器是否处理，得到100响应之后再发送post请求。

2. 2开头：表示成功接收请求，并且已经**完成整个处理过程**。

3. 3开头：重定向。

    - 301：永久重定向
    - 302：临时重定向

4. 4开头：客户端的请求有错误。

    - 400：客户端请求语法有问题
    - 403：禁止访问
    - 404：找不到资源

5. 5开头：服务器端出现错误。

    - 500：服务器内部错误
    - 503：服务不可用
    - 504：访问超时，服务器作为代理

✓ http与https的区别

http运行在tcp之上，使用明文进行传输，是无状态的（客户端和服务端都无法验证对方的身份），端口号是80；

https是http的基础上加了SSL(Secure Socket Layer)，在传输的过程中会进行加密和认证，端口号是443。资源消耗比http大。

✓ https工作原理

> SSL/TLS协议+http协议 = https协议
>
> SSL出现在会话层

1. 客户端发出请求

   客户端先向服务器发出加密通信的请求，并生成一个随机数，这被叫做ClientHello请求。

2. 服务器回应

   服务器收到客户端请求后，确定加密算法以及生成一个随机数，向客户端发出回应，把证书发给客户端，这叫做SeverHello。

3. 客户端回应

   客户端收到服务器回应以后，首先会验证服务器证书，如果证书可靠，则生成第三个随机数，接着取出证书里面的公钥对新生成随机数（pre-master key）进行加密，然后发给服务器。

4. 服务器的最后回应

   服务器收到客户端的加密后的随机数，利用私钥进行解密，此时客户端和服务器都拥有了三个随机数，然后两者根据约定好的加密算法生成本次会话所用的"会话密钥"。然后，服务器向客户端发送下面信息。

   编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。

   服务器握手结束通知，表示服务器的握手阶段已经结束。

至此，整个握手阶段全部结束。接下来，客户端与服务器进入加密通信，就完全是使用普通的HTTP协议，只不过用"会话密钥"加密内容，此步骤是对称加密。

> 证书包含哪些内容？证书的发布机构、证书的有效期、公钥等。
>
> 客户端如何检测数字证书是合法的？
>
> 首先客户端读取证书中的**发布机构**，然后会在**操作系统或浏览器**内置的受信任的发布机构中去找该机构的证书，如果找到，则**取出该机构证书中的公钥**，然后对服务器发送的证书**进行验签**，看有没有该证书是不是合法的，有没有被修改过。
>
> 【注意】CA机构在签发证书的时候，都会使用自己的私钥对证书进行签名

中间人攻击

中间人攻击是指客户端的请求被中间人拦截，然后中间人假装客户端向服务器发起请求。之后客户端的ssl握手以及请求都跟中间人进行，而中间人也可以跟服务端进行ssl握手跟请求。

使用中间人攻击手段，必须要让客户端信任中间人的证书，如果客户端不信任，则这种攻击手段也无法发挥作用。

✓ SSL握手的目的

保证客户端与服务器之间密钥的交换是安全的。握手期间使用的加密算法是非对称加密，通信期间使用的是对称加密。

✓ URI 和 URL 的区别是什么?

- **URI**：是统一资源**标识符**，是一个**资源的标识**。
- **URL**：是统一资源**定位符**，是一个**资源的路径**。URL 是 URI 的一个子集，它不仅唯一标识资源，还可以提供资源的定位信息。