# 计算机网络面试题

以下面试题，基于网络整理，和自己编辑。具体参考的文章，会在文末给出所有的链接。

如果胖友有自己的疑问，欢迎在星球提问，我们一起整理吊吊的【网络】面试题的大保健。

而题目的难度，尽量按照从容易到困难的顺序，逐步下去。

因为网络是个很大的话题，所以本文以常见的 TCP 和 HTTP 题目为主，例如：

- TCP 三次握手、四次挥手是什么？
- HTTP1.0、HTTP1.1、HTTP2 等的区别？
- ...

**注意，经常问的问题，会在前面加【重要】标识。**

# 网络体系结构

强烈推荐阅读两篇文章：

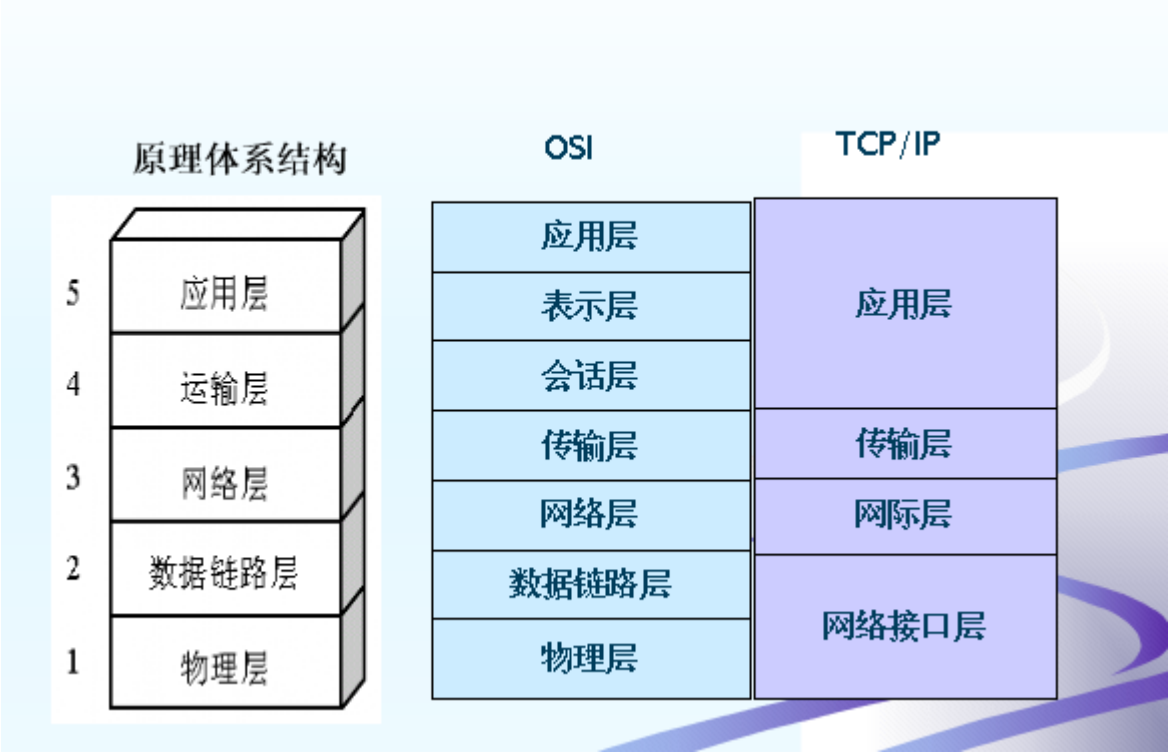
> ：可以后续回过头来看。

- 《计算机网络体系结构综述（上）》
- 《计算机网络体系结构综述（下）》

简单一瞥：

- **OSI 七层体系结构**具有概念清楚、理论完整的特点，是一个理论上的国际标准，但却不是事实上的国际标准。
- 具有简单易用特点的 **TCP/IP 四层体系结构**则是事实上的标准。
- 需要指出的是，**五层体系结构**虽然综合了 OSI 和 TCP/IP 的优点，但其只是为了学术学习研究而提出的，没有具体的实际意义。

- 三者结构示意图如下所示：




# 简述 OSI 七层体系结构

> ：比较复杂，至少要说出有哪些层。

自上而下是：

- 应用层（数据）：确定进程之间通信的性质以满足用户需要以及提供网络与用户应用
- 表示层（数据）：主要解决用户信息的语法表示问题，如加密解密
- 会话层（数据）：提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制，如服务器验证用户登录便是由会话层完成的
- 传输层（段）：实现网络不同主机上用户进程之间的数据通信，可靠 与不可靠的传输，传输层的错误检测，流量控制等
- 网络层（包）：提供逻辑地址（IP）、选路，数据从源端到目的端的 传输
- 数据链路层（帧）：将上层数据封装成帧，用MAC地址访问媒介，错误检测与修正
- 物理层（比特流）：设备之间比特流的传输，物理接口，电气特性等

详细可以看《计算机网络体系结构综述（下）》的「二. OSI 七层体系结构简述」小节。

OSI 七层体系结构

# 简述 TCP/IP 四层体系结构

> ：比较复杂，至少要说出有哪些层。

自上而下是：

> 实际上，如果我们把一些 RPC 框架的分层套到 TCP/IP 四层体系结构，也是可以的。跳到《精尽 Dubbo 面试题》的「Dubbo 框架的分层设计」问题瞅瞅噢。

- 应用层

> HTTP、TELNET、FTP、SMTP

- 传输层

> TCP、UDP

- 网络层

> IP、ICMP

- 数据接口

> PPP

详细可以看 《计算机网络体系结构综述（下）》 的 「三. TCP/IP 四层体系结构」 小节。

# 知道各个层使用的是哪个数据交换设备?

> 这个问题，了解即可。

- 网关：应用层、传输层。

> 网关在传输层上以实现网络互连，是最复杂的网络互连设备，仅用于两个高层协议不同的网络互连。
>
> 网关的结构也和路由器类似，不同的是互连层。网关既可以用于广域网互连，也可以用于局域网互连。

- 【重点】路由器：网络层

> 路由选择、存储转发

- 【重点】交换机：数据链路层、网络层

> 识别数据包中的 MAC 地址信息，根据 MAC 地址进行转发，并将这些 MAC 地址与对应的端口记录在自己内部的一个地址表中。

- 网桥：数据链路层

> 将两个 LAN 连起来，根据 MAC 地址来转发帧。

- 集线器（Hub）：物理层

> 纯硬件设备，主要用来连接计算机等网络终端。

- 中继器：物理层

> 在比特级别对网络信号进行再生和重定时，从而使得它们能够在网络上传输更长的距离。

**📣 交换机是什么?**

在计算机网络系统中，交换机是针对共享工作模式的弱点而推出的。交换机拥有一条高带宽的背部总线和内部交换矩阵。交换机的所有的端口都挂接在这条背部总线上，当控制电路收到数据包以后，处理端口会查找内存中的地址对照表以确定目的 MAC（网卡的硬件地址）的 NIC（网卡）挂接在哪个端口上，通过内部交换矩阵迅速将数据包传送到目的端口。**目的 MAC 若不存在，交换机才广播到所有的端口，接收端口回应后交换机会“学习”新的地址，并把它添加入内部地址表中。**

交换机工作于 OSI 参考模型的第二层，即数据链路层。交换机内部的 CPU 会在每个端口成功连接时，通过 ARP 协议学习它的 MAC 地址，保存成一张 ARP 表。在今后的通讯中，发往该 MAC 地址的数据包将仅送往其对应的端口，而不是所有的端口。因此，交换机可用于划分数据链路层广播，即冲突域；但它不能划分网络层广播，即广播域。

**📣 路由器是什么?**

路由器（Router），是一种计算机网络设备，提供了路由与转发两种重要机制，可以决定数据包从来源端到目的端所经过的路由路径（host 到 host 之间的传输路径），这个过程称为**路由**；将路由器输入端的数据包移送至适当的路由器输出端(在路由器内部进行)，这称为**转发**。所以，路由器的一个作用是连通不同的网络，另一个作用是选择信息传送的线路。

路由工作在 OSI 模型的第三层 —— 即网络层，例如 IP 协议。当然，这也是路由器与交换器的**差别**，路由器是属于 OSI 第三层的产品，交换器是 OSI 第二层的产品(这里特指二层交换机)。

**📣 常见的路由选择协议，以及它们的区别?**

常见的路由选择协议有：RIP 协议、OSPF 协议。

- RIP 协议：底层是贝尔曼福特算法，它选择路由的度量标准（metric)是跳数，最大跳数是 15 跳。如果大于 15 跳，它就会丢弃数据包。
- OSPF 协议：底层是迪杰斯特拉算法，是链路状态路由选择协议，它选择路由的度量标准是带宽，延迟。

**📣 什么是网关设备?**

这个我们就不在本文中多写，感兴趣的胖友，可以看看 《维基百科 —— 网关》 文章。

**📣 详细说明 Keepalived 的故障切换工作原理?**

这种故障切换，是通过 **VRRP** 协议来实现的。

- 主节点会按一定的时间间隔发送心跳信息的广播包，告诉备节点自己的存活状态信息。
- 当主节点发生故障时，备节点在一段时间内就收到广播包，从而判断主节点出现故障，因此会调用自身的接管程序来接管主节点的 IP 资源及服务。
- 当主节点恢复时，备节点会主动释放资源，恢复到接管前的状态，从而来实现主备故障切换

例如，MySQL 基于 Keepalived 实现高可用。详细的，可以看看 《Keepalived + MySQL实现高可用》 。

# IP

# IP 地址的分类?

IP 地址是指互联网协议地址，是 IP 协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。

IP 地址编址方案将IP地址空间划分为 A、B、C、D、E 五类，其中 A、B、C 是基本类，D、E 类作为多播和保留使用，为特殊地址。

每个 IP 地址包括两个标识码（ID），即网络 ID 和主机 ID 。同一个物理网络上的所有主机都使用同一个网络 ID，网络上的一个主机（包括网络上工作站，服务器和路由器等）有一个主机 ID 与其对应。A~E 类地址的特点如下：

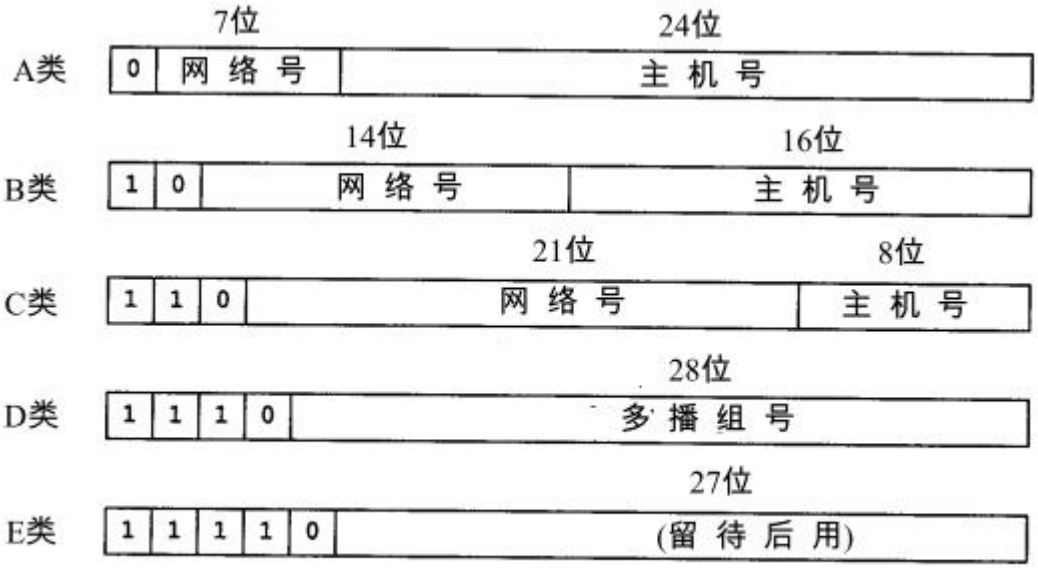



图1-5 五类互联网地址

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

详细的，可以看看 《IP 地址分类（A类 B类 C类 D类 E类）》 文章。

**IP 地址与物理地址的区别?**

- 物理地址(MAC 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。
- 其中 ARP 协议用于 IP 地址与物理地址的对应。

详细的，可以看看 《即生瑜，何生亮 —— MAC 地址与 IP 地址》 文章。

# 网络层的 ARP 协议工作原理?

网络层的 ARP 协议完成了 IP 地址与物理地址的映射。

- 首先，每台主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址的对应关系。
- 当源主机需要将一个数据包要发送到目的主机时，会首先检查自己 ARP 列表中是否存在该 IP 地址对应的 MAC 地址：
  - 如果有，就直接将数据包发送到这个 MAC 地址。
  - 如果没有，就向本地网段发起一个ARP请求的广播包，查询此目的主机对应的 MAC 地址。此 ARP 请求数据包里包括源主机的 IP 地址、硬件地址、以及目的主机的 IP 地址。网络中所有的主机收到这个 ARP 请求后，会检查数据包中的目的 IP 是否和自己的 I P地址一致。
    - 如果不相同，就忽略此数据包。
    - 如果相同，该主机首先将发送端的 MAC 地址和 IP 地址添加到自己的 ARP 列表中(如果 ARP 表中已经存在该 IP 的信息，则将其覆盖)，然后给源主机发送一个 ARP 响应数据包，告诉对方自己是它需要查找的 MAC 地址。

    - 源主机收到这个 ARP 响应数据包后，将得到的目的主机的 IP 地址和 MAC 地址添加到自己的 ARP 列表中，并利用此信息开始数据的传输。
    - 如果源主机一直没有收到 ARP 响应数据包，表示 ARP 查询失败。

> 注意，在 OSI 模型中 ARP 协议属于链路层；而在 TCP/IP 模型中，ARP 协议属于网络层。

# 如何划分子网、超网?

**📣 如何划分子网?**

> ：可选了解。

划分子网（变长子网掩码 VLSM）：划分子网的方法是从网络的主机号借用若干位作为子网号 subnet-id ，与此同时主机号也减少相应位数（总位数 32 位不变）。

由此两级 IP 地址可变为三级 IP 地址：`IP地址 ::= {<网络号>,<子网号>,<主机号>}` ，划分子网只是把 IP 地址的主机号这部分进行再划分，并不改变 IP 地址原来的网络号。

**📣 如何划分超网?**

> ：可选了解。

构造超网（无分类编址 CIDR）：CIDR 消除了传统的 A 类、B 类和 C 类地址以及划分子网的概念，把 32 位的 IP 地址划分为两个部分。

例如：`128.14.35.7/20` 是某个 CIDR 地址块中的一个地址，其前 20 位是网络前缀（用下划线表示的部分），后面的 12 位为主机号

**📣 子网掩码的作用?**

子网掩码只有一个作用，就是将某个 IP 地址划分成**网络地址**和**主机地址**两部分。

用于子网掩码的位数，决定于可能的子网数目和每个子网的主机数目。

# 什么是单播、组播(多播)、广播、任播?

> ：这个问题，一般面试应该不问，主要是为了大家扩充下知识面吧。

- **单播**(unicast): 是指封包在计算机网络的传输中，目的地址为单一目标的一种传输方式。它是现今网络应用最为广泛，通常所使用的网络协议或服务大多采用单播传输，例如一切基于TCP的协议。
- **组播**(multicast): 也叫多播， 多点广播或群播。 指把信息同时传递给一组目的地址。它使用策略是最高效的，因为消息在每条网络链路上只需传递一次，而且只有在链路分叉的时候，消息才会被复制。
- **广播**(broadcast):是指封包在计算机网络中传输时，目的地址为网络中所有设备的一种传输方式。实际上，这里所说的“所有设备”也是限定在一个范围之中，称为“广播域”。
- **任播**(anycast):是一种网络寻址和路由的策略，使得资料可以根据路由拓朴来决定送到“最近”或“最好”的目的地。

感兴趣的胖友，可以详细看 《单播，组播(多播)，广播以及任播》 文章。

# 区别 IPv4 和 IPv6 ?

- 我们大多数人使用的是第二代互联网 IPv4 技术，它的最大问题是网络地址资源有限，从理论上讲能编址 1600 万个网络、链接 40 亿台主机。而根据相关数据，全球 IPv4 的 IP 地址已经即将用完。

- 而 IPv6 是作为 IETF 设计的用于替代现行版本 IP 协议(IPv4)的下一代 IP 协议，其 IPV6 地址长度为 12 8位，地址空间增大了 2^98 次方倍，几乎可以说是用之不竭的。所以随着 IPv4 不足，支持 IPv6 的网络势必会增长。

# ICMP

> ：这个小节，可以快速看。因为，面试不一定问的很多。

## ICMP 协议的主要功能?

用于在 IP 主机、路由器之间传递控制消息。

如下图所示：

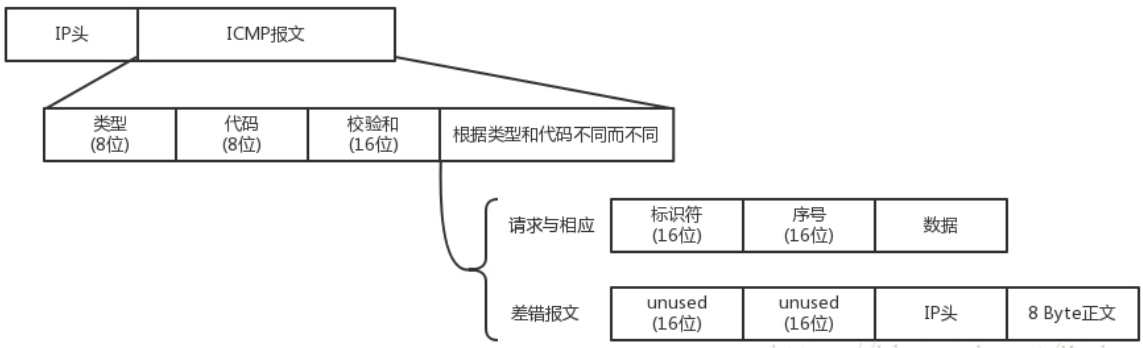
| ICMP 报文 | 类型的值 | ICMP 报文的类型 |
| --- | --- | --- |
| 差错报告报文 | 3 | 终点不可达 |
| | 4 | 源点抑制 |
| | 11 | 时间超过 |
| | 12 | 参数问题 |
| | 5 | 改变路由 |
| 询问报文 | 8 或0 | 回送请求或回答 |
| | 13 或14 | 时间戳请求或回答 |

## 简述一下 ping 的原理?

一般在网络不通的时候，大家会用 ping 测一下网络是否通畅。

ping 是基于 ICMP 协议工作的。ICMP 全称 Internet Control Message Protocol ，就是互联网**控制**报文协议。这里的关键词是“控制”，那具体是怎么控制的呢？ 网络包在异常负责的网络环境中传输时，会遇到各种问题，当遇到问题时，要传出消息，报告情况，这样才能调整传输策略。

ICMP 报文是封装在 IP 包里面的。因为传输指令的时候，肯定需要源地址和目标地址。如下图：






## 什么是 Traceroute ?

> Traceroute 是构建在 ICMP 协议之上的应用。

Traceroute ，是用来侦测主机到目的主机之间所经路由情况的重要工具，也是最便利的工具。

> 前面说到，尽管 ping 工具也可以进行侦测，但是，因为 IP 头的限制，ping 不能完全的记录下所经过的路由器。所以Traceroute 正好就填补了这个缺憾。

Traceroute 的**原理**是非常非常的有意思。

> - 它受到目的主机的 IP 后，首先给目的主机发送一个 TTL=1（还记得 TTL 是什么吗？）的 UDP(后面就知道 UDP是什么了)数据包，而经过的第一个路由器收到这个数据包以后，就自动把 TTL 减1，而 TTL 变为 0 以后，路由器就把这个包给抛弃了，并同时产生 一个主机不可达的 ICMP 数据报给主机。
> - 主机收到这个数据报以后再发一个 TTL=2 的 UDP 数据报给目的主机，然后刺激第二个路由器给主机发 ICMP 数据 报。如此往复直到到达目的主机。
>
> 这样，traceroute 就拿到了所有的路由器 IP 。从而避开了 IP 头只能记录有限路由 IP 的问题。

有人要问，我怎么知道 UDP 到没到达目的主机呢?

> 这就涉及一个技巧的问题，TCP 和 UDP 协议有一个端口号定义，而普通的网络程序只监控少数的几个号码较小的端口，比如说 80、23 等等。而 traceroute 发送的是端口号 >30000(真变态) 的 UDP 包，所以到达目的主机的时候，目的主机只能发送一个端口不可达的 ICMP 数据报给主机。主机接到这个报告以后就知道，目标主机到了。

😺 很多情况下，在我们 ping 不通目标地址时，会尝试使用 traceroute 命令，看看是否在中间哪个 IP 无法访问。

# TCP

## TCP 是什么?

TCP(Transmission Control Protocol)，传输控制协议，是一种面向连接的、可靠的、基于字节流的传输层通信协议。主要特点如下：

- TCP 是面向连接的。

  > 就好像打电话一样，通话前需要先拨号建立连接，通话结束后要挂机释放连接

- 每一条 TCP 连接只能有两个端点，每一条TCP连接只能是点对点的（一对一）。
- TCP 提供可靠交付的服务。通过TCP连接传送的数据，无差错、不丢失、不重复、并且按序到达。
- TCP 提供全双工通信。TCP 允许通信双方的应用进程在任何时候都能发送数据。TCP 连接的两端都设有发送缓存和接收缓存，用来临时存放双方通信的数据。
- 面向字节流。

  > TCP 中的“流”（Stream），指的是流入进程或从进程流出的字节序列。
  >
  > “面向字节流”的含义是：虽然应用程序和 TCP 的交互是一次一个数据块（大小不等），但 TCP 把应用程序交下来的数据仅仅看成是一连串的无结构的字节流。

## TCP 对应的应用层协议?

- FTP： 定义了文件传输协议，使用 21 端口。常说某某计算机开了 FTP 服务便是启动了文件传输服务。下载文件，上传主页，都要用到 FTP 服务。

- Telnet： 它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于 DOS 模式下的通信服务。如以前的 BBS 是纯字符界面的，支持 BBS 的服务器将 23 端口打开，对外提供服务。
- 邮箱
    - SMTP： 定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置-中常看到有这么 SMTP 端口设置这个栏，服务器开放的是 25 号端口。
    - POP3： 它是和 SMTP 对应，POP3 用于接收邮件。通常情况下，POP3 协议所用的是 110 端口。也是说，只要你有相应的使用 POP3 协议的程序（例如 Foxmail 或 Outlook），就可以不以 Web 方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是 163 邮箱就没有必要先进入网易网站，再进入自己的邮箱来收信）。
- HTTP： 从 Web 服务器传输超文本到本地浏览器的传送协议。

# TCP 头部是怎么样的?

- 《通俗大白话来理解 TCP 协议的三次握手和四次分手》 的 「TCP头部」 小节。
- 《TCP 协议的学习 （二）TCP 头部信息》

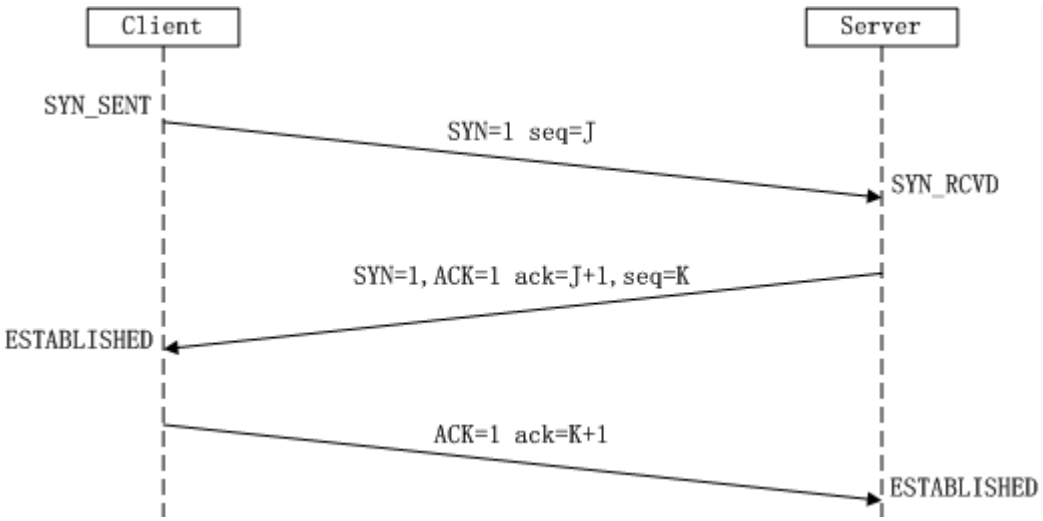
# 【重要】什么是 TCP 三次握手?

三次握手，简单来说，就是：

TCP 三次握手的漫画

- 发送方：我要和你建立链接?
- 接收方：你真的要和我建立链接么?
- 发送方：我真的要和你建立链接，成功。

详细来说，步骤如下：




- 第一次握手：Client 将标志位 `SYN=1` ，随机产生一个值 `seq=J` ，并将该数据包发送给 Server 。此时，Client 进入SYN_SENT 状态，等待 Server 确认。
- 第二次握手：Server 收到数据包后由标志位 `SYN=1` 知道Client请求建立连接，Server 将标志位 `SYN` 和 `ACK` 都置为 1， `ack=J+1` ，随机产生一个值 `seq=K` ，并将该数据包发送给 Client 以确认连接请求，Server 进入 `SYN_RCVD` 状态。此时，Server 进入 SYC_RCVD 状态。

- 第三次握手：Client 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 Server 。此时，Client 进入 ESTABLISHED 状态。
  - Server 检查 `ack` 是否为 `K+1` ，`ACK` 是否为 1 ，如果正确则连接建立成功。此时 Server 进入 ESTABLISHED 状态，完成三次握手，随后 Client 与 Server 之间可以开始传输数据了。
- 仔细看来，Client 会发起两次数据包，分别是 `SYNC` 和 `ACK` ；Server 会发起一次数据包，包含 `SYNC` 和 `ACK` 。也就是说，三次握手的过程中，Client 和 Server 互相做了一次 `SYNC` 和 `ACK` 。

**📣 为什么 TCP 连接需要三次握手，两次不可以么，为什么?**

为了防止**已失效的连接请求**报文突然又传送到了服务端，因而产生错误。

> 客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达 Server 。
>
> - 若不采用"三次握手"，那么只要 Server 发出确认数据包，新的连接就建立了。由于 Client 此时并未发出建立连接的请求，所以其不会理睬 Server 的确认，也不与 Server 通信；而这时 Server 一直在等待 Client 的请求，这样 Server 就白白浪费了一定的资源。
> - 若采用"三次握手"，在这种情况下，由于 Server 端没有收到来自客户端的确认，则就会知道 Client 并没有要求建立请求，就不会建立连接。

在 《通俗大白话来理解 TCP 协议的三次握手和四次挥手》 中，搜 "为什么要三次握手" 关键字，也有非常好的解答。

- **这就很明白了，防止了服务器端的一直等待而浪费资源。**

**📣 客户端不断进行请求链接会怎样?**

服务器端准备为每个请求创建一个链接，并向其发送确认报文，然后等待客户端进行确认后创建。如果此时客户端一直不确认，会造成 SYN 攻击，即:

> SYN 攻击，英文为 SYN Flood ，是一种典型的 DoS/DDoS 攻击。

- 1、客户端向服务端发送请求连接数据包。
- 2、服务端向客户端发送确认数据包。
- 3、客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认。

  > 这是这一步！！！

🚀 如何检测 SYN 攻击？检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源 IP 地址是随机的，基本上可以断定这是一次 SYN 攻击。在 Linux/Unix 上可以使用系统自带的 netstat 命令来检测 SYN 攻击。

🚀 怎么解决 SYN 攻击呢？答案是**只能预防**，没有彻底根治的办法，除非不使用 TCP 。方式如下:

- 1、限制同时打开 SYN 半链接的数目

  > ：是不是很像我们常用的"限流"。

- 2、缩短 SYN 半链接的 Timeout 时间

  > ：是不是很像我们常用的"超时"。

- 3、关闭不必要的服务。

  > ：酱紫，这个服务就不会被 SYN 攻击连接。

- 4、增加最大半连接数。
- 5、过滤网关防护。
- 6、SYN cookie技术。

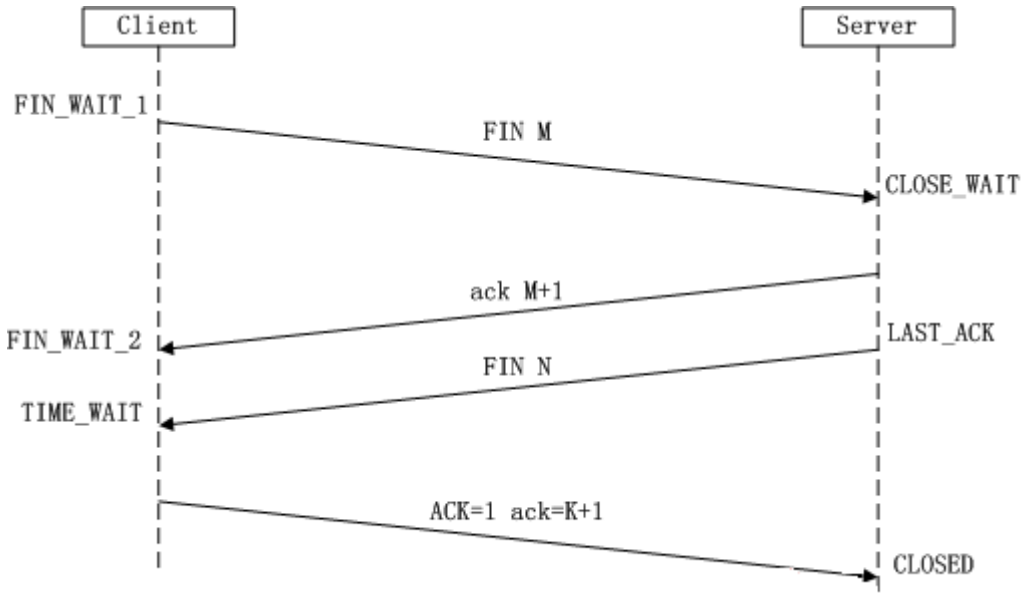
> ：有一点点重新“设计” TCP 的味道，或者可以理解成对 TCP 的增强。

# 【重要】什么是 TCP 四次挥手?

四次挥手，简单来说，就是：

- 发送方：我要和你断开连接！
- 接收方：好的，断吧。
- 接收方：我也要和你断开连接！
- 发送方：好的，断吧。

详细来说，步骤如下：




> 如下使用 Client 和 Server 的方式，仅仅是为了方便，也是可以从 Server 向 Client 发起。

- 第一次挥手：Client 发送一个 `FIN=M` ，用来关闭 Client 到 Server 的数据传送。此时，Client 进入 FIN_WAIT_1 状态。
- 第二次挥手，Server 收到 `FIN` 后，发送一个 `ACK` 给 Client ，确认序号为 `M+1` （与 `SYN` 相同，一个 `FIN` 占用一个序号）。此时，Server 进入 CLOSE_WAIT 状态。注意，TCP 链接处于**半关闭**状态，即客户端已经没有要发送的数据了，但服务端若发送数据，则客户端仍要接收。
- 第三次挥手，Server 发送一个 `FIN=N` ，用来关闭 Server 到 Client 的数据传送。此时 Server 进入 LAST_ACK 状态。
- 第四次挥手，Client 收到 `FIN` 后，此时 Client 进入 TIME_WAIT 状态。接着，Client 发送一个 `ACK` 给 Server ，确认序号为 `N+1` 。Server 接收到后，此时 Server 进入 CLOSED 状态，完成四次挥手。

**📣 为什么要四次挥手?**

TCP 协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。**TCP 是全双工模式**，这就意味着：

- 当主机 1 发出 `FIN` 报文段时，只是表示主机 1 已经没有数据要发送了，主机 1 告诉主机 2，它的数据已经全部发送完毕了；**但是，这个时候主机 1 还是可以接受来自主机 2 的数据；**当主机 2 返回 `ACK` 报文段时，表示它已经知道主机 1 没有数据发送了，但是主机 2 还是可以发送数据到主机 1 的。

  > ：因为主机 2 此时可能还有数据想要发送给主机 1，所以挥手不能像握手只有三次，而是多了那么“一次”！

- 当主机 2 也发送了 `FIN` 报文段时，这个时候就表示主机 2 也没有数据要发送了，就会告诉主机 1，我也没有数据要发送了，之后彼此就会愉快的中断这次 TCP 连接。

  > ：我们把四次挥手，理解成一次和平的挥手~

如果要正确的理解四次的原理，就需要了解四次挥手过程中的状态变化。

> 主动方=发送方；被动方=接收方。
>
> 状态前面的（主动方）（被动方），表示该状态属于谁。

- （主动方）FIN_WAIT_1：这个状态要好好解释一下，其实 FIN_WAIT_1 和 FIN_WAIT_2 状态的真正含义都是表示等待对方的 `FIN` 报文。而这两种状态的区别是：
  - FIN_WAIT_1 状态实际上是当 Socket 在 ESTABLISHED 状态时，它想主动关闭连接，向对方发送了 `FIN` 报文，此时该 Socket 即进入到 FIN_WAIT_1 状态。
  - 而当对方回应 `ACK` 报文后，则进入到 FIN_WAIT_2 状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应 `ACK` 报文。所以， FIN_WAIT_1 状态一般是比较难见到的，而 FIN_WAIT_2 状态还有时常常可以用 netstat 看到。
- （主动方）FIN_WAIT_2：上面已经详细解释了这种状态，实际上 FIN_WAIT_2 状态下的 Socket，表示半连接，也即有一方要求 close 连接，但另外还告诉对方，我暂时还有点数据需要传送给你( `ACK` 信息)，稍后再关闭连接。
- （被动方）CLOSE_WAIT：这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方 close 一个 Socket 后发送 `FIN` 报文给自己，你系统毫无疑问地会回应一个 `ACK` 报文给对方，此时则进入到 CLOSE_WAIT 状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以 close 这个 Socket，发送 `FIN` 报文给对方，也即关闭连接。所以你在 CLOSE_WAIT 状态下，需要完成的事情是等待你去关闭连接。
- （被动方）LAST_ACK：这个状态还是比较容易好理解的，它是被动关闭一方在发送 `FIN` 报文后，最后等待对方的 `ACK` 报文。当收到 `ACK` 报文后，也即可以进入到 CLOSED 可用状态了。
- （主动方）TIME_WAIT：表示收到了对方的 `FIN` 报文，并发送出了 `ACK` 报文，就等 2MSL 后即可回到 CLOSED 可用状态了。如果 FIN_WAIT_1 状态下，收到了对方同时带 `FIN` 标志和 `ACK` 标志的报文时，可以直接进入到 TIME_WAIT 状态，而无须经过 FIN_WAIT_2 状态。

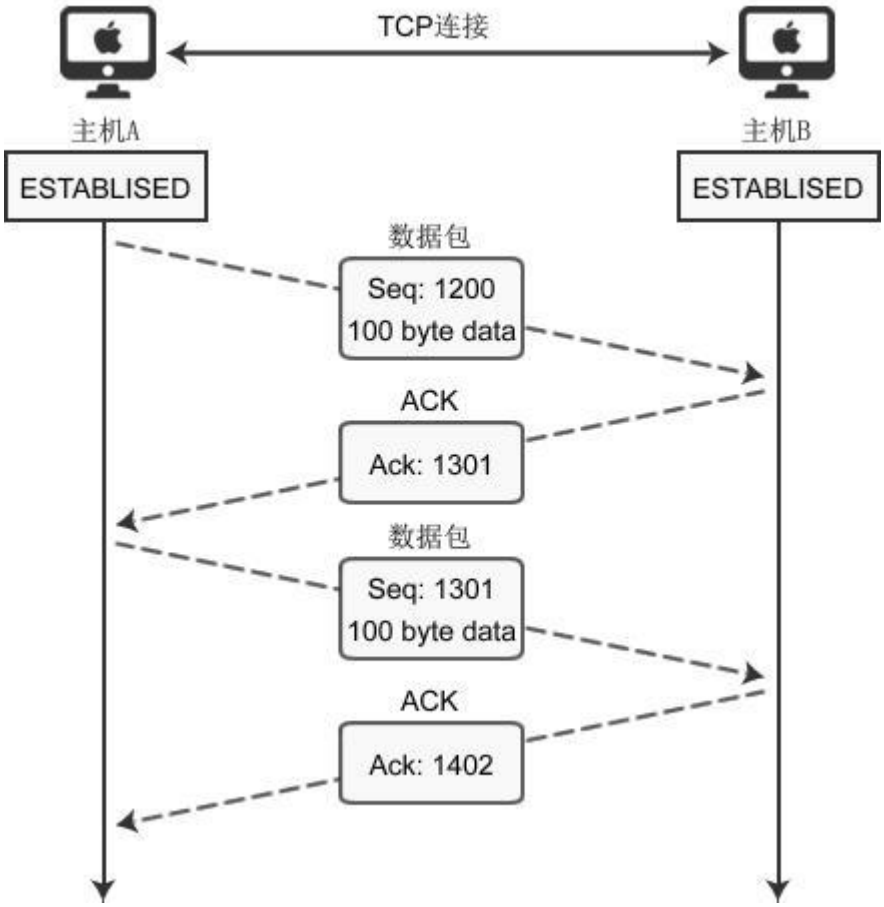
  > **为何一定要等 2MSL？**
  >
  > 如果不等，释放的端口可能会重连刚断开的服务器端口，这样依然存活在网络里的老的 TCP 报文可能与新 TCP 连接报文冲突，造成数据冲突，为避免此种情况，需要耐心等待网络老的 TCP 连接的活跃报文全部死翘翘，2MSL 时间可以满足这个需求（尽管非常保守）！
  >
  > 更多，可以看看知乎 《为什么 TCP 4 次挥手时等待为 2MSL？》 的讨论。

- CLOSED：表示连接中断。

另外，关于 TIME_WAIT 和 CLOSE_WAIT 状态的区别，胖友可以在细看下 《TIME_WAIT 和 CLOSE_WAIT 状态区别》 。

# 【重要】TCP 数据如何传输?

建立连接后，两台主机就可以相互传输数据了。如下图所示：




- 上图给出了主机 A 分 2 次（分 2 个数据包）向主机 B 传递 200 字节的过程。
- 首先，主机 A 通过 1 个数据包发送 100 个字节的数据，数据包的 `Seq` 号设置为 1200 。主机 B 为了确认这一点，向主机 A 发送 `ACK` 包，并将 `Ack` 号设置为 1301 。
    - 为了保证数据准确到达，目标机器在收到数据包（包括 `SYN` 包、 `FIN` 包、普通数据包等）包后必须立即回传 `ACK` 包，这样发送方才能确认数据传输成功。
    - 此时 `Ack` 号为 1301 而不是 1201，原因在于 `Ack` 号的增量为传输的数据字节数。假设每次 `Ack` 号不加传输的字节数，这样虽然可以确认数据包的传输，但无法明确 100 字节全部正确传递还是丢失了一部分，比如只传递了 80 字节。因此按如下的公式确认：

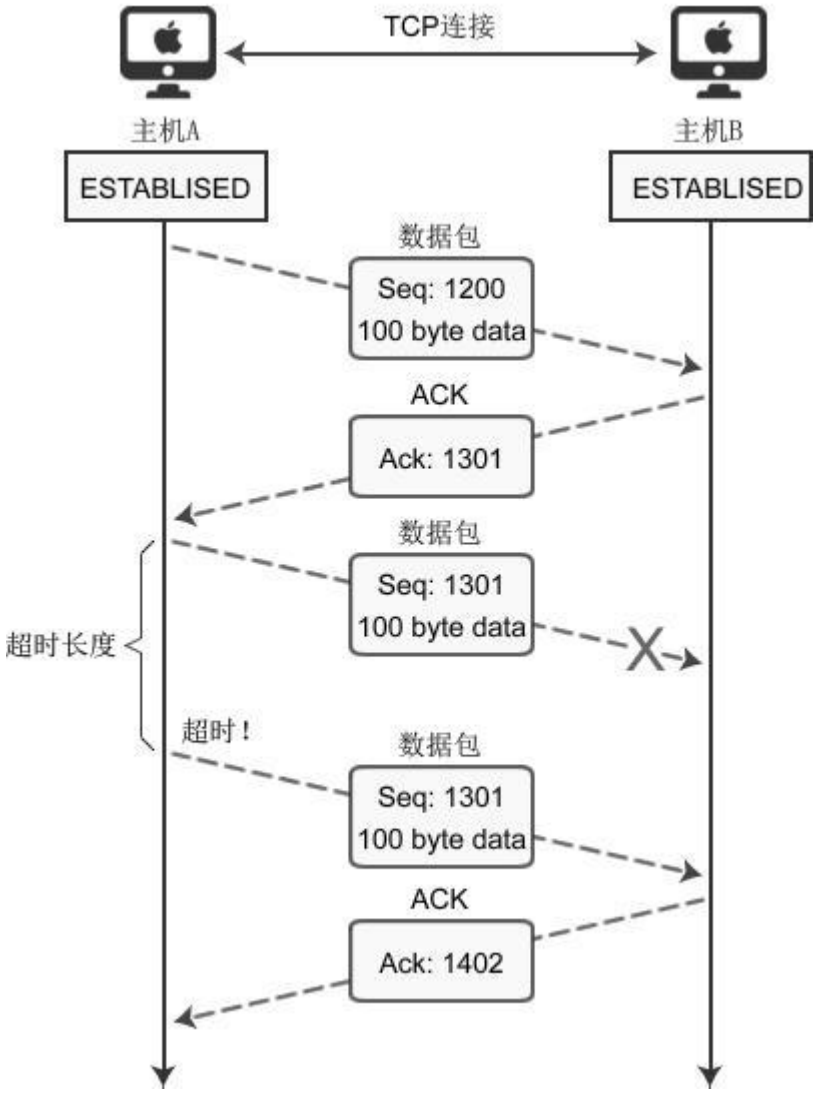
      `Ack` 号： `Ack` 号 = `Seq` 号 + 传递的字节数 + 1 。
        - 与三次握手协议相同，最后加 1 是为了告诉对方要传递的 `Seq` 号。

OK，让我们重新来看下 TCP 的整个过程。如下图所示：TCP 过程

**📣 TCP 数据传输丢失怎么办?**

> ：这个问题，也可以改成提问，什么是 TCP 重传。

因为各种原因，TCP 数据包可能存在丢失的情况，TCP 会进行数据重传。如下图所示：




- 上图表示通过 `Seq` 1301 数据包向主机 B 传递 100 字节的数据，但中间发生了错误，主机 B 未收到。经过一段时间后，主机 A 仍未收到对于 `Seq` 1301 的 `ACK` 确认，因此尝试重传数据。为了完成数据包的重传，TCP 套接字每次发送数据包时都会**启动定时器**，如果在一定时间内没有收到目标机器传回的 `ACK` 包，那么定时器超时，数据包会重传。上图演示的是数据包丢失的情况，也会有 `ACK` 包丢失的情况，一样会重传。
- 重传超时时间(RTO，Retransmission Time Out)

  > 这个值太大了会导致不必要的等待，太小会导致不必要的重传，理论上最好是网络 RTT 时间，但又受制于网络距离与瞬态时延变化，所以实际上使用自适应的动态算法（例如 Jacobson 算法和 Karn 算法等）来确定超时时间。
  >
  > 往返时间（RTT，Round-Trip Time）表示从发送端发送数据开始，到发送端收到来自接收端的 `ACK` 确认包（接收端收到数据后便立即确认），总共经历的时延。

- 重传次数

  > TCP 数据包重传次数，根据系统设置的不同而有所区别。有些系统，一个数据包只会被重传 3 次，如果重传 3 次后还未收到该数据包的 `ACK` 确认，就不再尝试重传。但有些要求很高的业务系统，会不断地重传丢失的数据包，以尽最大可能保证业务数据的正常交互。
  >
  > 最后需要说明的是，发送端只有在收到对方的 `ACK` 确认包后，才会清空输出缓冲区中的数据。

**ps：TCP 数据传输的过程，和 MQ Broker 投递消息给 Consumer 是一样的，只有在 Consumer Ack 确认消息已经消费，该消息才不会再被投递给 Consumer 。**

另外，也推荐阅读 《网络基本功（九）：细说TCP重传》 。

# 【重要】什么是 TCP 滑动窗口?

在看 TCP 滑动窗口的概念之前，我们先来看看它出现的**背景**?

> 将 TCP 与 UDP 这样的简单传输协议区分开来的是，它传输数据的质量。TCP 对于发送数据进行跟踪，这种数据管理需要协议有以下两大关键功能：
>
> - 可靠性：保证数据确实到达目的地。如果未到达，能够发现并重传。
> - 数据流控：管理数据的发送速率，以使接收设备不致于过载。
>
> 要完成这些任务，整个协议操作是围绕**滑动窗口** + **确认机制**来进行的。因此，理解了滑动窗口，也就是理解了 TCP 。

那么，到底什么是 TCP 滑动窗口呢?

> 滑动窗口协议，是传输层进行流控的一种措施，接收方通过通告发送方自己的窗口大小，从而控制发送方的发送速度，从而达到防止发送方发送速度过快而导致自己被淹没的目的。
>
> TCP 的滑动窗口解决了端到端的流量控制问题，允许接受方对传输进行限制，直到它拥有足够的缓冲空间来容纳更多的数据。

可能这么描述之后，胖友会有点懵逼，那么建议看下面三篇文章，耐心~

- 《TCP 滑动窗口控制流量的原理》
  > 比较易懂的一篇文章。
- 《网络基本功（八）：细说 TCP 滑动窗口》
  > 更为详细的一篇文章。
- 《TCP 协议的滑动窗口具体是怎样控制流量的？》
  > 知乎上的讨论，重点看「wuxinliulei」和「安静的木小昊」的回答。特别是后者的，回答很生动形象。

# TCP 协议如何来保证传输的可靠性?

TCP 提供一种面向连接的、可靠的字节流服务。其中，面向连接意味着两个使用 TCP 的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个 TCP 连接。

- 在一个 TCP 连接中，仅有两方进行彼此通信。
- 而字节流服务意味着两个应用程序通过 TCP 链接交换 8bit 字节构成的字节流，TCP 不在字节流中插入记录标识符。

对于可靠性，TCP 通过以下方式进行保证：

- 数据包校验：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时 TCP 发送数据端超时后会重发数据。
- 对失序数据包重排序：既然 TCP 报文段作为 IP 数据报来传输，而 IP 数据报的到达可能会失序，因此 TC P报文段的到达也可能会失序。TCP 将对失序数据进行重新排序，然后才交给应用层。
- 丢弃重复数据：对于重复数据，能够丢弃重复数据。
- 应答机制：当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒。

- 超时重发：当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。
- 流量控制：TCP 连接的每一方都有固定大小的缓冲空间。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据，这可以防止较快主机致使较慢主机的缓冲区溢出，这就是流量控制。TCP 使用的流量控制协议是可变大小的**滑动窗口协议**。

# 什么是 TCP 拥堵?

计算机网络中的带宽、交换结点中的缓存及处理机等都是网络的资源。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就会变坏，这种情况就叫做**拥塞**。

**怎么解决 TCP 拥堵?**

通过拥塞控制来解决。拥堵控制，就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。注意，拥塞控制和流量控制不同，前者是一个**全局性**的过程，而后者指**点对点**通信量的控制。

拥塞控制的方法主要有以下四种：

- 1、慢开始。
- 2、拥塞避免。
- 3、快重传。
- 4、快恢复。

**1）慢开始**

不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。

**2）拥塞避免**

拥塞避免算法，让拥塞窗口缓慢增长，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1 ，而不是加倍，这样拥塞窗口按线性规律缓慢增长。

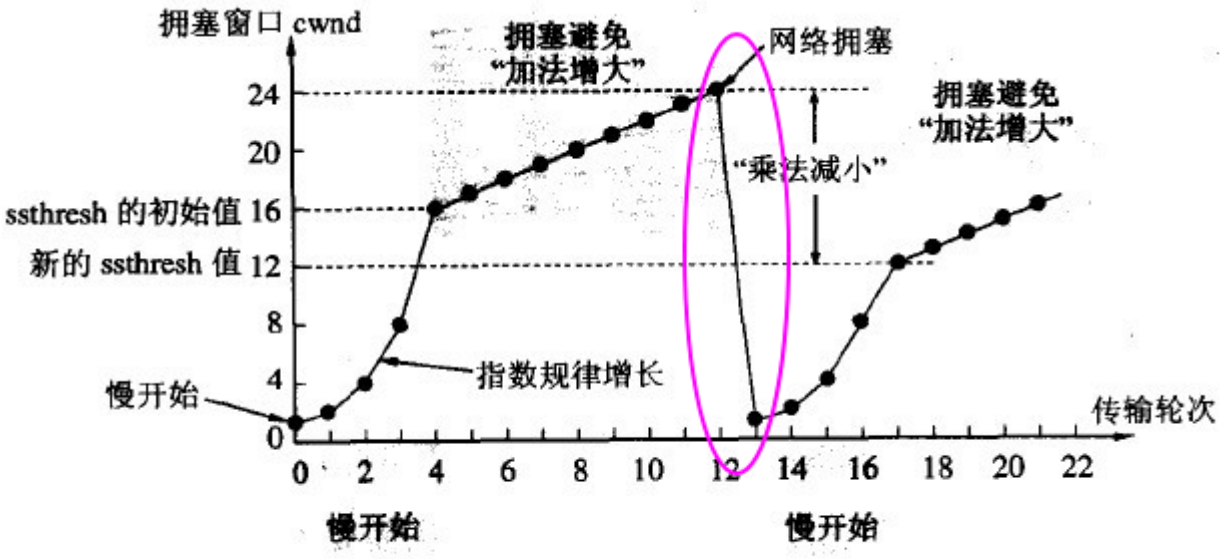



图 5-25　慢开始和拥塞避免算法的实现举例 net/sicofield

**3）快重传**

快重传，要求接收方在收到一个**失序的报文段**后就立即发出**重复确认**（为的是使发送方及早知道有报文段没有到达对方），而不要等到自己发送数据时捎带确认。

快重传算法规定，发送方只要一连收到三个重复确认，就应当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。

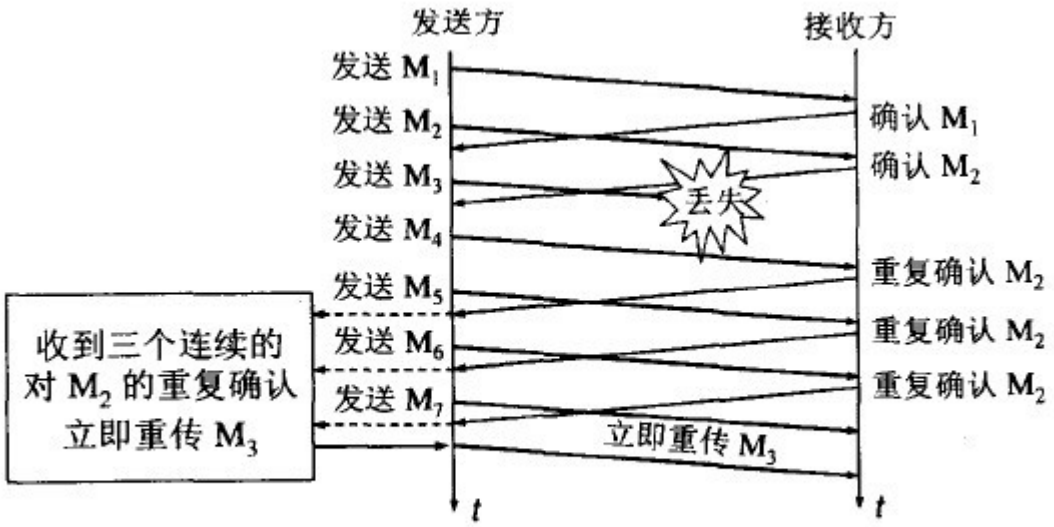



图 5-26　快重传的示意图



**4）快恢复**

快重传配合使用的还有快恢复算法，当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把 ssthresh 门限减半。

- 但是接下去并不执行慢开始算法：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。
- 所以此时不执行慢开始算法，而是将 cwnd 设置为 ssthresh 的大小，然后执行拥塞避免算法。

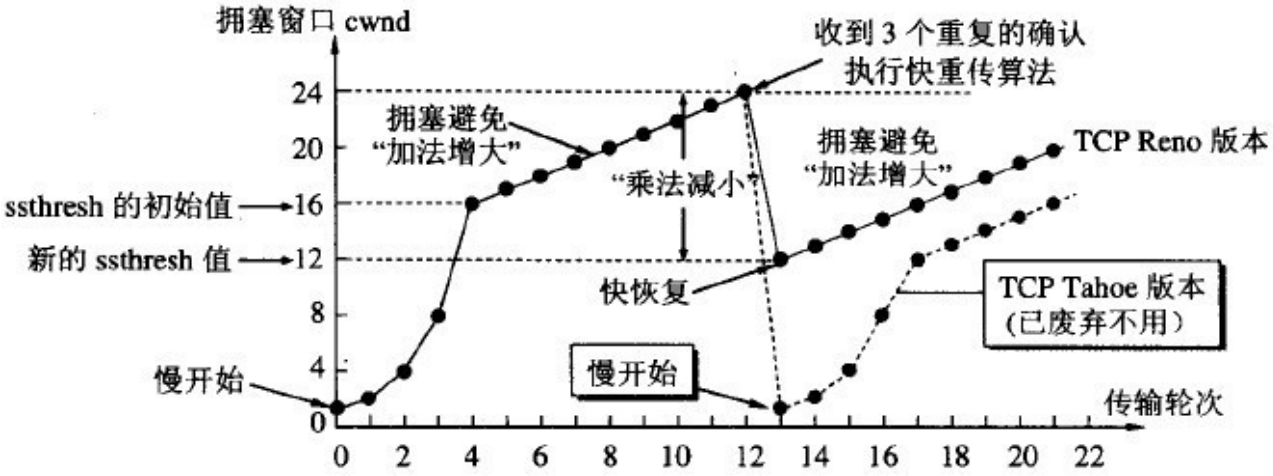



图 5-27　从连续收到三个重复的确认转入拥塞避免

# UDP

## UDP 是什么?

UDP（User Data Protocol，用户数据报协议），是与 TCP 相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去。

主要特点如下：

- UDP 是无连接的。
- UDP 使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态（这里面有许多参数）。
- 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 直播，实时视频会议等

- UDP 支持一对一、一对多、多对一和多对多的交互通信。
- UDP 的首部开销小，只有 8 个字节，比 TCP 的 20 个字节的首部要短。

## UDP 对应的应用层协议?

- DNS：用于域名解析服务，将域名地址转换为 IP 地址。DNS 用的是 53 号端口。
- SNMP：简单网络管理协议，使用 161 号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。
- TFTP(Trivial File Transfer Protocol)：简单文件传输协议，该协议在熟知端口 69 上使用 UDP 服务。

## 【重要】TCP 与 UDP 的区别

> 这个问题，上面在介绍 TCP 和 UDP 都提到了，就是做了整合哈。

TCP(Transmission Control Protocol)和 UDP(User Datagram Protocol) 协议属于传输层协议，它们之间的区别包括：

| 类型 | 特点 | | | 性能 | | 应用场景 | 首部字节 |
|---|---|---|---|---|---|---|---|
| | 是否面向连接 | 传输可靠性 | 传输形式 | 传输效率 | 所需资源 | | |
| TCP | 面向连接 | 可靠 | 字节流 | 慢 | 多 | 要求通信数据可靠<br>（如文件传输、邮件传输） | 20-60 |
| UDP | 无连接 | 不可靠 | 数据报文段 | 快 | 少 | 要求通信速度高<br>（如域名转换） | 8个字节<br>（由4个字段组成） |

- TCP 是面向连接的；UDP 是无连接的。
- TCP 是可靠的；UDP 是不可靠的。
- TCP 只支持点对点通信；UDP 支持一对一、一对多、多对一、多对多的通信模式。
- TCP 是面向字节流的；UDP 是面向报文的。
- TCP 有拥塞控制机制；UDP 没有拥塞控制，适合媒体通信。

- TCP 首部开销(20 个字节)，比 UDP 的首部开销(8 个字节)要大。

📣 **为什么 TCP 叫数据流模式？ UDP 叫数据报模式?**

所谓的**“流模式”**，是指**TCP 发送端发送几次数据和接收端接收几次数据是没有必然联系的。**

- 比如你通过 TCP 连接给另一端发送数据，你只调用了一次 write ，发送了 100 个字节，但是对方可以分 10 次收完，每次 10 个字节；你也可以调用 10 次 write ，每次 10 个字节，但是对方可以一次就收完。
- 原因：这是因为 TCP 是面向连接的，一个 Socket 中收到的数据都是由同一台主机发出，且有序地到达，所以每次读取多少数据都可以。

所谓的**“数据报模式”**，是指 **UDP 发送端调用了几次 write ， 接收端必须用相同次数的 read 读完。**

- UDP 是基于报文的，在接收的时候，每次最多只能读取一个报文，报文和报文是不会合并的，如果缓冲区小于报文长度，则多出的部分会被丢弃。
- 原因：这是因为 UDP 是无连接的，只要知道接收端的 IP 和端口，任何主机都可以向接收端发送数据。这时候，如果一次能读取超过一个报文的数据，则会乱套。

## UDP 报文的格式


- 16 位 * 4 = 64 位 = 8 字节。

# DNS

## DNS 是什么?

- 域名解析，www.xxx.com 转换成 IP ，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。
- DNS 协议运行在 UDP 协议之上，使用端口号 53 。

## 主机解析域名的顺序?

1. 浏览器缓存
2. 找本机的 hosts 文件
3. 路由缓存
4. 找 DNS 服务器(本地域名、顶级域名、根域名)

    - 迭代查询
    - 递归查询

## DNS 使用什么协议?

> 参见《DNS使用的是 TCP 协议还是 UDP 协议》文章。

既使用 TCP 又使用 UDP 。

- 区域传送时使用 TCP 协议。
    - 辅域名服务器会定时（一般时 3 小时）向主域名服务器进行查询以便了解数据是否有变动。如有变动，则会执行一次区域传送，进行数据同步。区域传送将使用 TCP 而不是 UDP ，因为数据同步传送的数据量比一个请求和应答的数据量要多得多。
    - TCP 是一种可靠的连接，保证了数据的准确性。
- 域名解析时使用 UDP 协议。
    - 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。
        > UDP 报文的最大长度为 512 字节。
    - 不用经过 TCP 三次握手，这样 DNS 服务器负载更低，响应更快。虽然从理论上说，客户端也可以指定向 DNS 服务器查询的时候使用 TCP ，但事实上，很多 DNS 服务器进行配置的时候，仅支持 UDP 查询包。

# HTTP

## HTTP 是什么?

HTTP 协议，是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

主要特点如下：

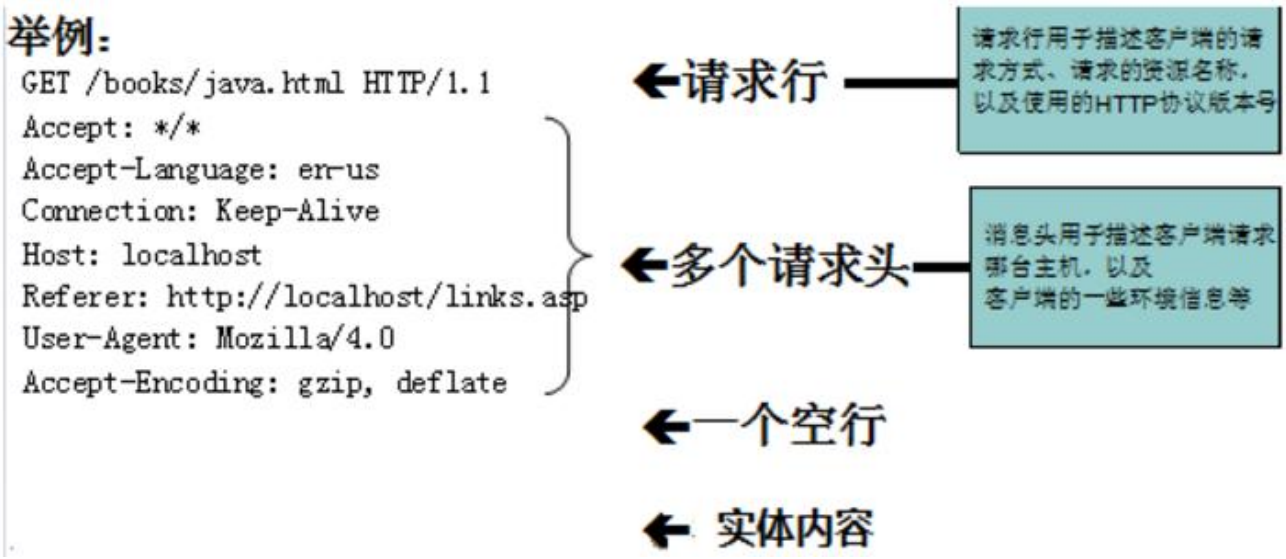
- 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST 等等。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- 数据格式灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由Content-Type 加以标记。
- 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。
    > 主要指的是不使用 Keep-Alive 机制的情况下。
- 无状态：HTTP 协议是无状态协议。无状态，是指协议对于事务处理没有记忆能力。无状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。
    > 无状态，所以更容易做服务的扩容，支撑更大的访问量。
- 支持 B/S 及 C/S 模式。

> 另外，HTTP 协议已经不仅仅使用在浏览器上。在前后端分离的架构中，又或者微服务架构的内部通信中，HTTP 因为其数据格式的通用性，和语言无关，被大规模使用。

# HTTP 基本格式

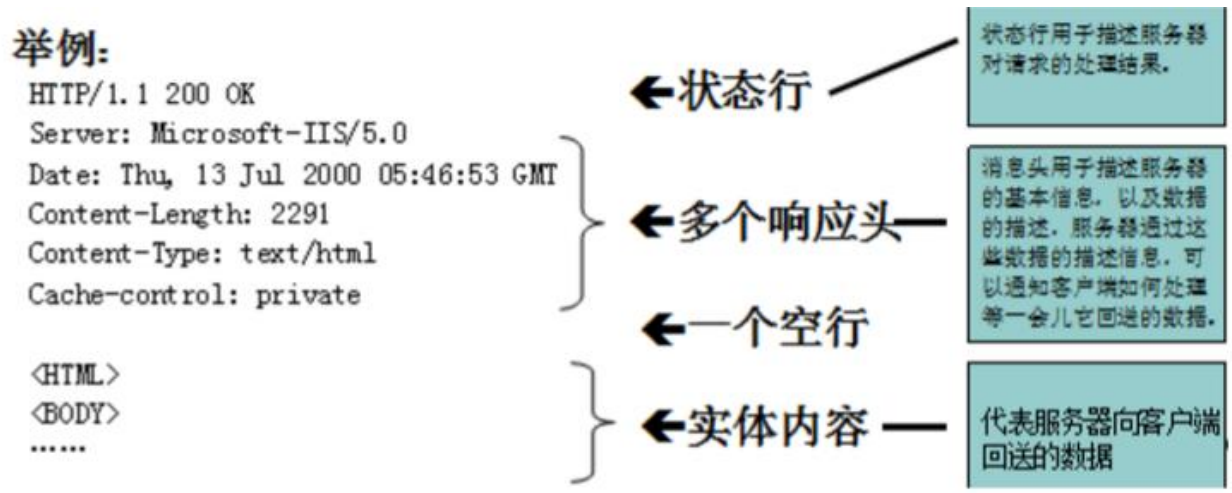
> ：详细的，可以看看 《猫哥网络编程系列：详解 BAT 面试题》 文章。

**HTTP 请求格式**




- 请求行：用来说明请求类型，要访问的资源以及所使用的 HTTP 版本。
- 请求头部：紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息从第二行起为请求头部。
  - HOST ，将指出请求的目的地。
  - User-Agent ，服务器端和客户端脚本都能访问它,它是浏览器类型检测逻辑的重要基础。该信息由你的浏览器来定义，并且在每个请求中自动发送等等
  - ...
- 空行：请求头部后面的空行是必须的。
- 请求数据：也叫主体，可以添加任意的其他数据。

**HTTP 响应格式**



- 状态行：由 HTTP 协议版本号、状态码、状态消息三部分组成。
- 消息报头：用来说明客户端要使用的一些附加信息。
- 空行：消息报头后面的空行是必须的。
- 响应正文：服务器返回给客户端的文本信息。

**📣 URI 和 URL 的区别?**

见 《URI 和 URL 的区别》 文章。

# HTTP 协议包括哪些请求?

- GET: 对服务器资源的简单请求。
- POST: 用于发送包含用户提交数据的请求。
- HEAD：类似于 GET 请求，不过返回的响应中没有具体内容，用于获取报头。
- PUT：传说中请求文档的一个版本。
- DELETE：发出一个删除指定文档的请求。
- TRACE：发送一个请求副本，以跟踪其处理进程。
- OPTIONS：返回所有可用的方法，检查服务器支持哪些方法。
- CONNECT：用于 SSL 隧道的基于代理的请求。

**📣 GET 和 POST 的区别?**

| 请求方式 | 数据位置 | 明文密文 | 数据安全 | 长度限制 | 应用场景 |
| --- | --- | --- | --- | --- | --- |
| GET | HTTP 请求的 path 中 | 明文 | 不安全 | 长度较小，一般 2k | 查询数据 |
| POST | HTTP 请求 body 中 | 可明可密 | 安全 | 支持较大数据传输 | 修改数据 |

- GET 请求可被缓存；POST 请求不会被缓存。
- GET 请求可被收藏为书签；POST 不能被收藏为书签。
- 【非常有趣】参见 《99%的人理解错 HTTP 中 GET 与 POST 的区别》

> - 对于 GET 方式的请求，浏览器会把 HTTP header 和 data 一并发送出去，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）。
>
> 也就是说，GET 只需要汽车跑一趟就把货送到了，而 POS T得跑两趟，第一趟，先去和服务器打个招呼“嗨，我等下要送一批货来，你们打开门迎接我”，然后再回头把货送过去。
>
> ps：不过要注意，POST 具体发几次，也和浏览器的实现有关系。例如：Firefox 只发一次。 ps2：据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的 TCP 在验证数据包完整性上，有非常大的优点。

# HTTP 有哪些状态码?

- 1×× : 请求处理中，请求已被接受，正在处理
- 2×× : 请求成功，请求被成功处理
  - 200 OK // 客户端请求成功
- 3×× : 重定向，要完成请求必须进行进一步处理

    - 301 Moved Permanently // 永久重定向,使用域名跳转
    - 302 Found // 临时重定向,未登陆的用户访问用户中心重定向到登录页面
- 4×× : 客户端错误，请求不合法
    - 400 Bad Request // 客户端请求有语法错误，不能被服务器所理解
    - 401 Unauthorized // 请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
    - 403 Forbidden // 服务器收到请求，但是拒绝提供服务
    - 404 Not Found // 请求资源不存在，eg：输入了错误的 URL
- 5×× : 服务器端错误，服务器不能处理合法请求
    - 500 Internal Server Error // 服务器发生不可预期的错误
    - 503 Server Unavailable // 服务器当前不能处理客户端的请求，一段时间后可能恢复正常

完整的状态码列表，可以看看 《HTTP 状态码》 文章。

📣 **forward 和 redirect 的区别?**

- 直接转发方式（Forward），客户端和浏览器只发出一次请求，Servlet、HTML、JSP 或其它信息资源，由第二个信息资源响应该请求，在请求对象 request 中，保存的对象对于每个信息资源是共享的。
- 间接转发方式（Redirect），实际是两次 HTTP 请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。

详细的，请看 《请求转发（Forward）和重定向（Redirect）的区别》 。

📣 **HTTP 返回码中 301 与 302 的区别?**

301，302 都是 HTTP 状态的编码，都代表着某个 URL 发生了转移，不同之处在于：

- 301 redirect: 301 代表永久性转移(Permanently Moved)。
- 302 redirect: 302 代表暂时性转移(Temporarily Moved)。

详细的，请看 《HTTP 返回码中 301 与 302 的区别》 文章。

# HTTP、TCP、Socket 的关系是什么?

- TCP/IP 代表传输控制协议/网际协议，指的是一系列协议族。
- HTTP 本身就是一个协议，是从 Web 服务器传输超文本到本地浏览器的传送协议。
- Socket 是 TCP/IP 网络的 API，其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面。对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。

综上所述：

- 需要 IP 协议来连接网络
- TCP 是一种允许我们安全传输数据的机制，使用 TCP 协议来传输数据的 HTTP 是 Web 服务器和客户端使用的特殊协议。
- HTTP 基于 TCP 协议，所以可以使用 Socket 去建立一个 TCP 连接。

# Cookies 和 Session 的区别

- Session 在服务器端，Cookie 在客户端（浏览器）。

    > Session 默认被存在在服务器的一个文件里（不是内存）。

- Session 的运行依赖 sessionid，而 sessionid 是存在 Cookie 中的，也就是说，如果浏览器禁用了 Cookie，同时 session 也会失效。但是，可以通过其它方式实现，比如在 url 参数中传递 sessionid 。

- Session 可以放在文件、数据库、或内存中都可以。
- 【关键】**用户验证这种场合一般会用 Session 。**

# 【重要】一次完整的 HTTP 请求所经历的步骤

> 这里的客户端，更多指的是浏览器。

- 1、DNS 解析(通过访问的域名找出其 IP 地址，递归搜索)。
- 2、HTTP 请求，当输入一个请求时，建立一个 Socket 连接发起 TCP的 3 次握手。

  > 如果是 HTTPS 请求，会略微有不同。等到 HTTPS 小节，我们在来讲。

- 3.1、客户端向服务器发送请求命令（一般是 GET 或 POST 请求）。

  > 这个是补充内容，面试一般不用回答。
  >
  > 客户端的网络层不用关心应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，我不作过多的描述，无非就是通过查找路由表决定通过那个路径到达服务器。
  >
  > 客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定 IP 地址的 MAC 地址，然后发送 ARP 请求查找目的地址，如果得到回应后就可以使用 ARP 的请求应答交换的 IP 数据包现在就可以传输了，然后发送 IP 数据包到达服务器的地址。

- 3.2、客户端发送请求头信息和数据。
- 4.1、服务器发送应答头信息。
- 4.2、服务器向客户端发送数据。
- 5、服务器关闭 TCP 连接（4次挥手）。

  > 这里是否关闭 TCP 连接，也根据 HTTP Keep-Alive 机制有关。
  >
  > 同时，客户端也可以主动发起关闭 TCP 连接。

- 6、客户端根据返回的 HTML、CSS、JS 进行渲染。

如下是《图解HTTP》提供的图片：

| 过程 | 使用的协议 |
| --- | --- |
| 1. 浏览器查找域名的IP地址<br>(DNS查找过程：浏览器缓存、路由器缓存、DNS 缓存) | DNS：获取域名对应IP |
| 2. 浏览器向web服务器发送一个HTTP请求<br>(cookies会随着请求发送给服务器) | • TCP：与服务器建立TCP连接<br>• IP：建立TCP协议时，需要发送数据，发送数据在网络层使用IP协议<br>• OPSF：IP数据包在路由器之间，路由选择使用OPSF协议<br>• ARP：路由器在与服务器通信时，需要将ip地址转换为MAC地址，需要使用ARP协议<br>• HTTP：在TCP建立完成后，使用HTTP协议访问网页 |
| 3. 服务器处理请求<br>(请求 处理请求 & 它的参数、cookies、生成一个HTML响应) | |
| 4. 服务器发回一个HTML响应 | |
| 5. 浏览器开始显示HTML | |

# HTTP1.0 和 HTTP1.1 有什么区别?

主要是如下 8 点：

- 1、可扩展性
- 2、缓存
- 3、带宽优化

  > 带来了分块传输 。可能的话，面试也会问。

- 【最重要】4、长连接
- 5、消息传递
- 6、Host 头域
- 7、错误提示
- 8、内容协商

详细的每一点的说明，可以看 《HTTP1.0 与 HTTP1.1 的区别》 文章，特别是第 4 点【长连接】。

> HTTP1.1 支持长连接（PersistentConnection）和请求的流水线（Pipelining）。
>
> - 长连接（PersistentConnection）：处理在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟。在 HTTP1.1中 默认开启 `Connection: keep-alive` ，一定程度上弥补了 HTTP1.0 每次请求都要创建连接的缺点。
> - 请求的流水线（Pipelining）：HTTP1.1 还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分

> 出每次请求的响应内容，这样也显著地减少了整个下载过程所需要的时间。
>
> 推荐，在看看 《HTTP Keep-Alive 是什么？如何工作？》 文章。
>
> 关于这一点，可能演变的问题有：
>
> - HTTP 的长连接是什么意思？
> - HTTP Keep-Alive 机制是什么？
> - HTTP Keep-Alive 机制和 TCP Keep-Alive 有什么区别？

# SPDY 是什么？

> ：关于这个问题，了解就好。
>
> HTTP Working-Group 最终决定以 SPDY/2 为基础，开发 HTTP/2 。

2012 年，Google 如一声惊雷提出了 SPDY 的方案，优化了 HTTP1.X 的请求延迟，解决了 HTTP1.X 的安全性，具体如下：

- 1、降低延迟

  > 针对 HTTP 高延迟的问题，SPDY 优雅的采取了多路复用（multiplexing）。多路复用通过多个请求 Stream 共享一个 Tcp连 接的方式，解决了 HOL blocking 的问题，降低了延迟同时提高了带宽的利用率。

- 2、请求优先级（request prioritization）

  > 多路复用带来一个新的问题是，在连接共享的基础之上有可能会导致关键请求被阻塞。SPDY 允许给每个 request 设置优先级，这样重要的请求就会优先得到响应。
  >
  > 比如浏览器加载首页，首页的 html 内容应该优先展示，之后才是各种静态资源文件，脚本文件等加载，这样可以保证用户能第一时间看到网页内容。

- 3、header 压缩

  > 前面提到 HTTP1.x 的 header 很多时候都是重复多余的。选择合适的压缩算法可以减小包的大小和数量。

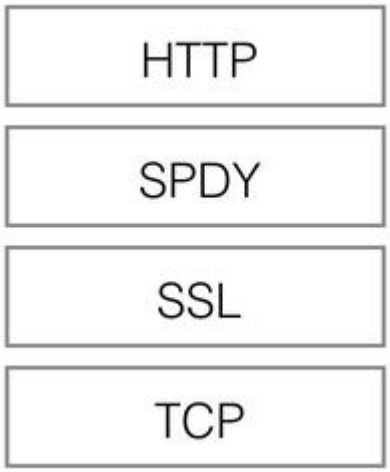
- 4、基于 HTTPS 的加密协议传输

  > 大大提高了传输数据的安全性。

- 5、服务端推送（server push）

  > 采用了 SPDY 的网页，例如我的网页有一个 `sytle.css` 的请求，在客户端收到 `sytle.css` 数据的同时，服务端会将 `sytle.js` 的文件推送给客户端。当客户端再次尝试获取 `sytle.js` 时就可以直接从缓存中获取到，不用再发请求了。
  >
  > ☺ 和我们理解的服务端推送，有点（非常）不一样哈。

🚀 SPDY 构成图如下：




- SPDY 位于 HTTP 之下，TCP 和 SSL 之上，这样可以轻松兼容老版本的 HTTP 协议(将 HTTP1.x 的内容封装成一种新的 frame 格式)，同时可以使用已有的 SSL 功能。

# HTTPS

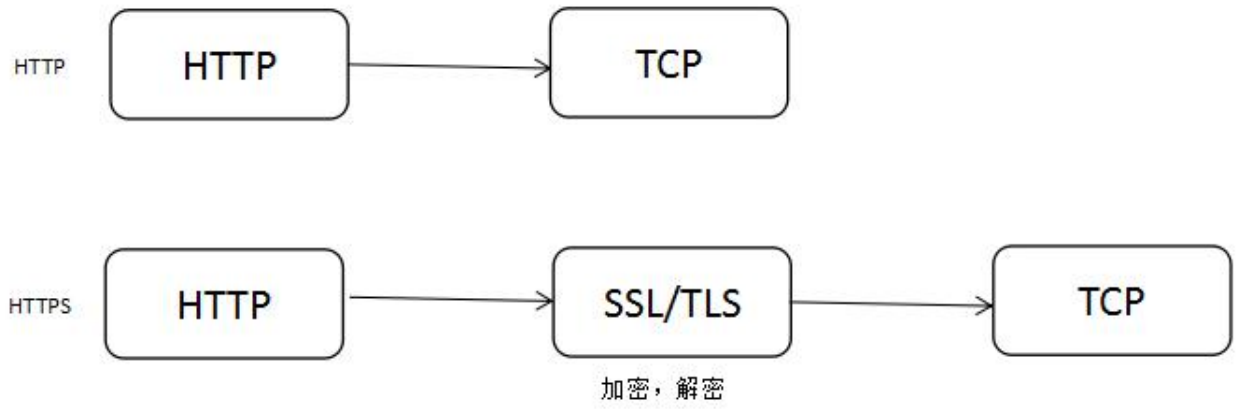
推荐先看下 《九个问题从入门到熟悉 HTTPS》 文章，写的很有趣~

另外，也看看 《SSL/TLS 双向认证(一) – SSL/TLS工作原理》 文章，写的更技术向~

下面的面试题的答案，我们会基于上述文章来整理。

## HTTPS 是什么?

HTTPS ，实际就是在 TCP 层与 HTTP 层之间加入了 SSL/TLS 来为上层的安全保驾护航，主要用到对称加密、非对称加密、证书，等技术进行客户端与服务器的数据加密传输，最终达到保证整个通信的安全性。




一句话概括：**HTTP + 加密 + 认证 + 完整性保护 = HTTPS** 。

**什么是 SSL 呢? 什么是 TLS 呢?**

官方定义，SSL 是安全套接层(secure sockets layer)；TLS 是 SSL 的继任者，叫传输层安全(transport layer security)。

它们存在的唯一目的就是保证上层通讯安全的一套机制。它的发展依次经历了下面几个时期，像手机软件升级一样，每次更新都添加或去除功能，比如引进新的加密算法，修改握手方式等。

- SSL1.0: 已废除
- SSL2.0: RFC6176 ， 已废除
- SSL3.0: RFC6101 ， 基本废除
- **TLS1.0**: RFC2246 ， 目前大都采用此种方式
- TLS1.1: RFC4346
- TLS1.2: RFC5246 ， 没有广泛使用
- TLS1.3: IETF 正在酝酿中

> ：为了下面描述方便，统一先叫 SSL 。

**SSL/TLS 协议作用?**

1. 认证用户和服务器，确保数据发送到正确的客户机和服务器。

   > 客户端必须避免中间人攻击，即除了真正的服务器，任何第三方都无法冒充服务器。

2. 加密数据以防止数据中途被窃取。
3. 维护数据的完整性，确保数据在传输过程中不被改变。

# HTTP 和 HTTPS 的区别?

- 端口不同：HTTP 与 HTTPS 使用不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

  > ：个人的想法，实际 HTTPS 也是可以使用 80 端口，但是考虑继续保持 HTTP 的兼容，只好退而求其次，使用 443 端口。

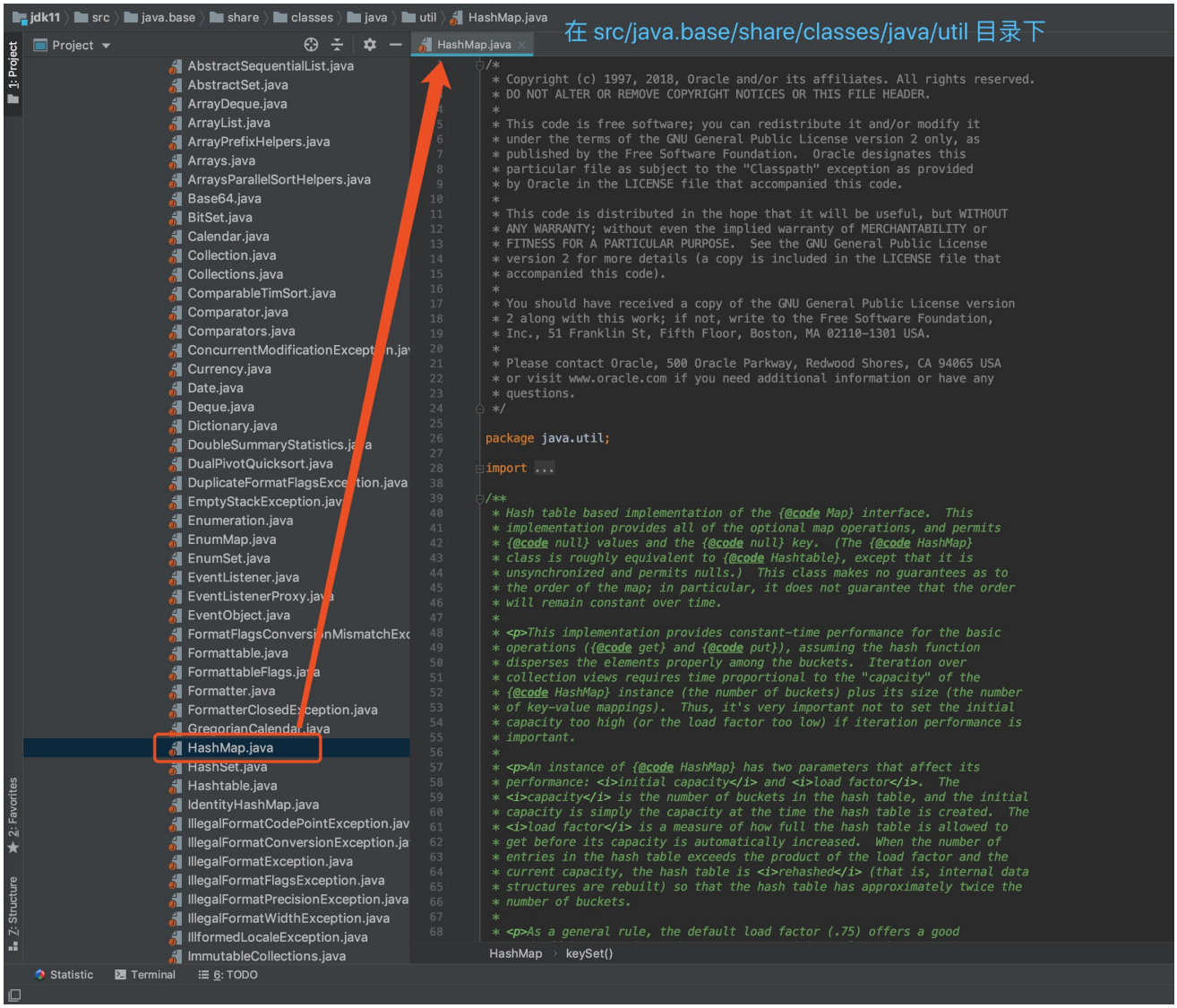
- 资源消耗：和 HTTP 通信相比，HTTPS 通信会由于加解密处理消耗更多的 CPU 和内存资源。
- 开销：HTTPS 通信需要证书，而证书一般需要向认证机构申请免费或者付费购买。

  > HTTPS 可以有效的防止运营商劫持，解决了防劫持的一个大问题。

# SSL 加密方式是什么?

> - 对称密钥加密，是指加密和解密使用同一个密钥的方式，这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方。
> - 非对称加密，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处理，对方接收到加密信息后，使用自己的私钥进行解密。

SSL 协议，即用到了对称加密也用到了非对称加密，如下图所示：




- 在建立传输链路时，SSL 首先对对称加密的密钥使用公钥进行非对称加密。

  > ：注意哟，这里 Server 返回给 Client 的不是公钥( `server.pub` )，而是 `server.crt` 。Client 需要使用 `ca.key` 从 `server.crt` 中解密出公钥( `server.pub` )。

- 链路建立好之后，SSL 对传输内容使用公钥( `server.pub` )对称加密。

**为什么公钥传输的步骤这么复杂呢?**

答案请看 《九个问题从入门到熟悉 HTTPS》 文章的如下问题：

- Q5: 那公钥怎么传输
- Q6: 你在逗我么。。。。
- Q7: 怎么知道证书有没有被篡改?
- Q8: 这样可以防止第三方冒充服务器么

也就是说，通过 CA 来保证。至于 `server.crt` 证书是怎么申请的呢？请看 《SSL/TLS 双向认证(一) - SSL/TLS工作原理》 文章的 「CA 的证书 ca.crt 和 SSL Server 的证书 server.crt 是什么关系呢？」 问题的解答。

> ：看这块，我已经要被绕晕了！！！耐心~胖友，理解后会很爽。

# 什么是单向认证、双向认证?

- 单向认证，指的是只有一个对象校验对端的证书合法性。

  > 通常都是 Client 来校验服务器的合法性。那么 Client 需要一个`ca.crt`，服务器需要`server.crt`和`server.key`。

- 双向认证，指的是相互校验，Server 需要校验每个 Client，Client 也需要校验服务器。

  > - Server 需要`server.key`、`server.crt`、`ca.crt`。
  > - Client 需要`client.key`、`client.crt`、`ca.crt`。

**📣 1）单向认证的过程?**

单向认证

- 1、客户端向服务端发送 SSL 协议版本号、加密算法种类、随机数等信息。
- 2、服务端给客户端返回 SSL 协议版本号、加密算法种类、随机数等信息，同时也返回服务器端的证书，即公钥证书。
- 3、客户端使用服务端返回的信息验证服务器的合法性，包括：
  - 证书是否过期。
  - 发型服务器证书的 CA 是否可靠。
  - 返回的公钥是否能正确解开返回证书中的数字签名。
  - 服务器证书上的域名是否和服务器的实际域名相匹配

    > 验证通过后，将继续进行通信；否则，终止通信。

  - 4、客户端向服务端发送自己所能支持的对称加密方案，供服务器端进行选择。
  - 5、服务器端在客户端提供的加密方案中选择加密程度最高的加密方式。
- 6、服务器将选择好的加密方案通过明文方式返回给客户端。
- 7、客户端接收到服务端返回的加密方式后，使用该加密方式生成产生随机码，用作通信过程中对称加密的密钥，使用服务端返回的公钥进行加密，将加密后的随机码发送至服务器。
- 8、服务器收到客户端返回的加密信息后，使用自己的私钥进行解密，获取对称加密密钥。

> **在接下来的会话中，服务器和客户端将会使用该密码进行对称加密，保证通信过程中信息的安全。**

**📣 2）双向认证的过程?**

双向认证和单向认证原理基本差不多，只是除了客户端需要认证服务端以外，增加了服务端对客户端的认证，具体过程如下：

双向认证

- 1、客户端向服务端发送 SSL 协议版本号、加密算法种类、随机数等信息。
- 2、服务端给客户端返回 SSL 协议版本号、加密算法种类、随机数等信息，同时也返回服务器端的证书，即公钥证书。
- 3、客户端使用服务端返回的信息验证服务器的合法性，包括：
  - 证书是否过期。
  - 发型服务器证书的 CA 是否可靠。
  - 返回的公钥是否能正确解开返回证书中的数字签名。
  - 服务器证书上的域名是否和服务器的实际域名相匹配

> 验证通过后，将继续进行通信；否则，终止通信。

    - **【新增】4、服务端要求客户端发送客户端的证书，客户端会将自己的证书发送至服务端。**
    - **【新增】5、验证客户端的证书，通过验证后，会获得客户端的公钥。**
    - 6、客户端向服务端发送自己所能支持的对称加密方案，供服务器端进行选择。
    - 7、服务器端在客户端提供的加密方案中选择加密程度最高的加密方式。
- 8、服务器将选择好的加密方案通过明文方式返回给客户端。
- 9、客户端接收到服务端返回的加密方式后，使用该加密方式生成产生随机码，用作通信过程中对称加密的密钥，使用服务端返回的公钥进行加密，将加密后的随机码发送至服务器。
- 10、服务器收到客户端返回的加密信息后，使用自己的私钥进行解密，获取对称加密密钥。

> **在接下来的会话中，服务器和客户端将会使用该密码进行对称加密，保证通信过程中信息的安全。**

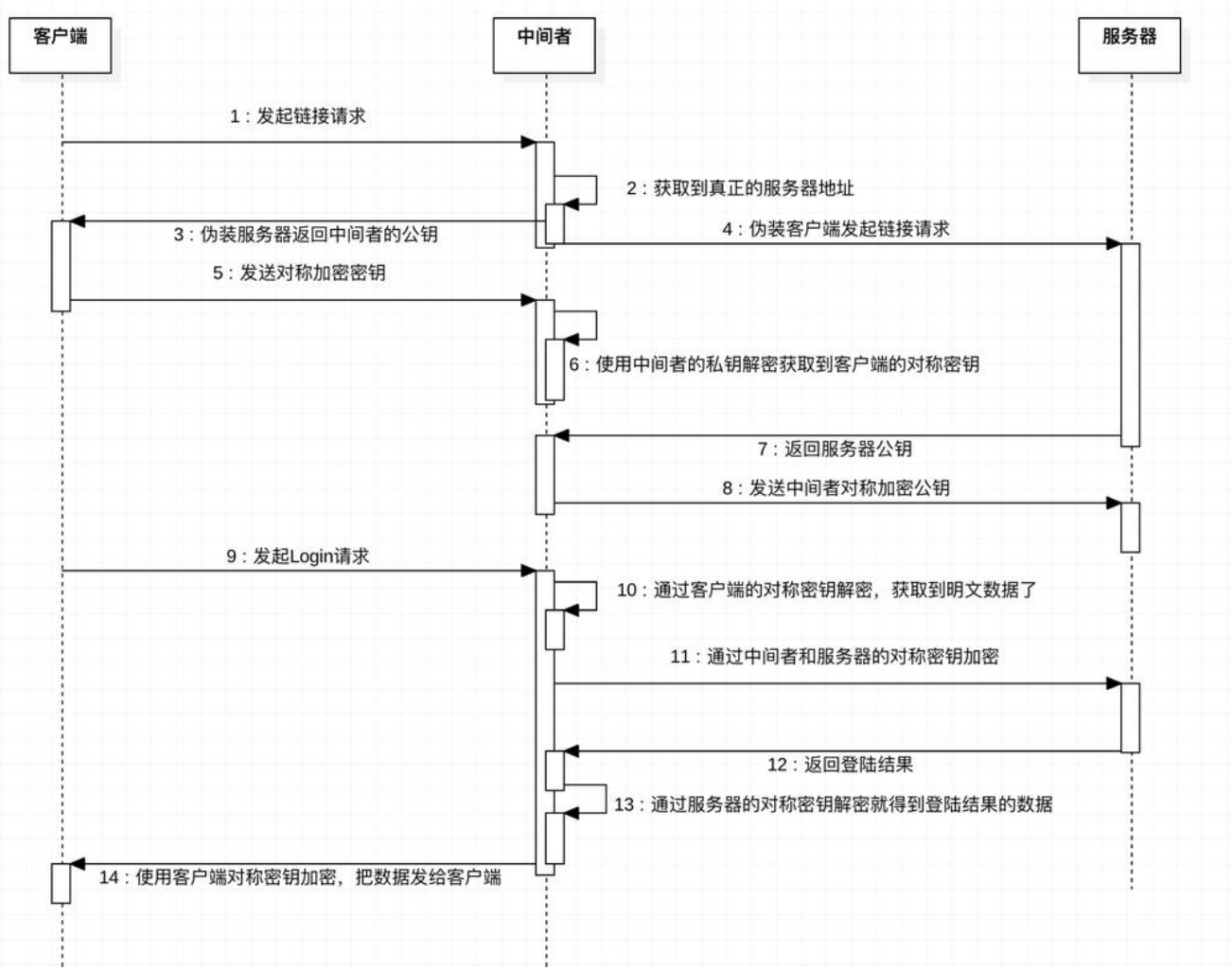
📣 **如何选择单向认证还是双向认证**

- 一般一个站点很多用户访问就用单向认证。
- 企业接口对接就用双向认证。

> 如果想要提高 APP 的安全级别，也可以考虑双向认证。因为，APP 天然方便放入客户端证书，从而提高安全级别。

# 为什么抓包工具还能抓到 HTTPS 数据包并解密成功呢

不是说HTTPS在网络中传输的是密文吗？这个问题就是**中间者攻击**（man in zhe middle）。



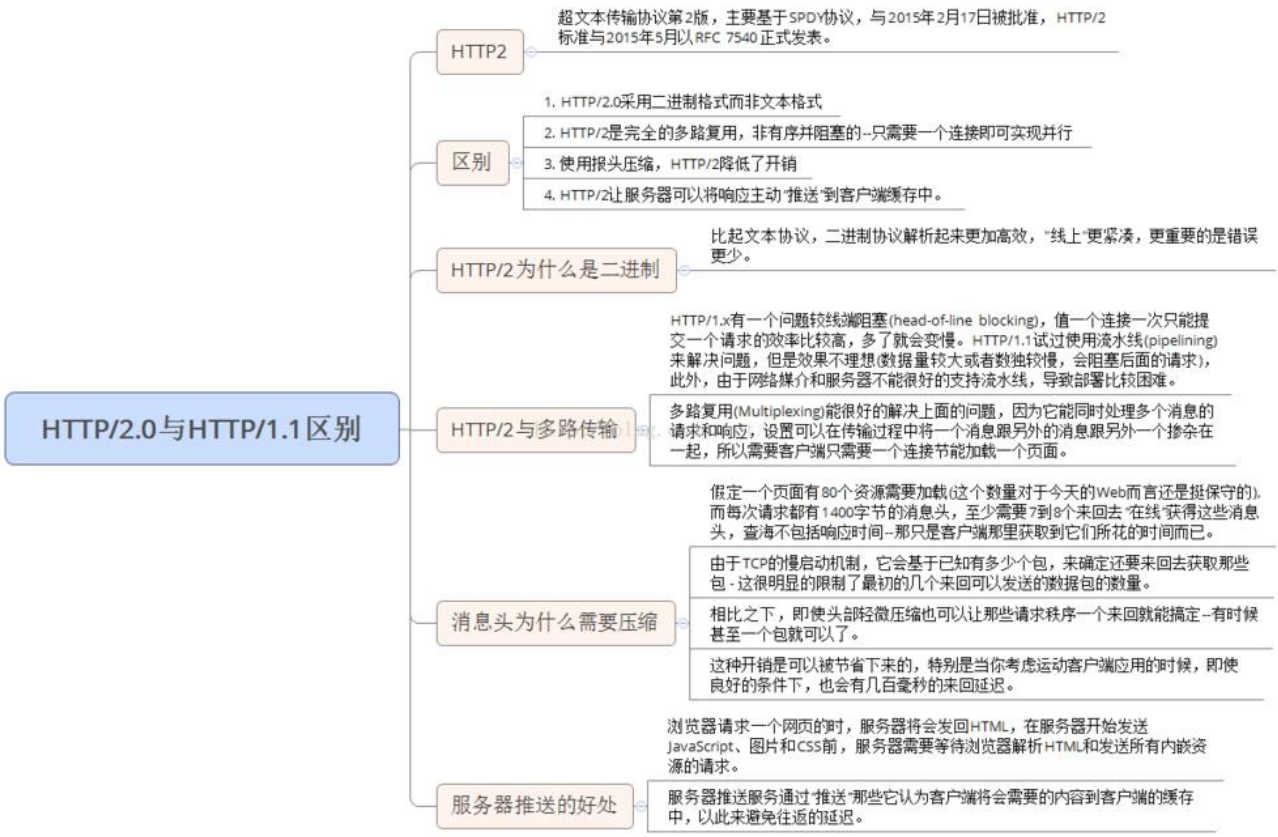

- 解决办法，就是 HTTPS 单向验证。在客户端中内置服务器公钥，在第三步服务器返回的公钥，除了验证公钥的有效性之外，再比对公钥是不是和内置的公钥一样，不一样说明被中间者攻击了，就断开链接不在请求了。
- 这个原理的前提是服务器的私钥没有泄露，客户端的代码不会被破解，道高一尺魔高一丈。信息安全就是在合理的范围内，选择比较合适的加密方法，没有绝对论，只有相对论。在某个范围内比较安全。

## HTTPS 握手会影响性能么?

TCP 有三次握手，再加上 HTTPS 的四次握手，影响肯定有，但是可以接受。

- 首先，HTTPS 肯定会更慢一点，时间主要花费在两组 SSL 之间的耗时和证书的读取验证上，对称算法的加解密时间几乎可以忽略不计。
- 而且如果不是首次握手，后续的请求并不需要完整的握手过程。客户端可以把上次的加密情况直接发送给服务器从而快速恢复，具体细节可以参考 《图解 SSL/TLS 协议》 。
- 除此以外，SSL 握手的时间并不是只能用来传递加密信息，还可以承担起客户端和服务器沟通 HTTP2 兼容情况的任务。因此从 HTTPS 切换到 HTTP2.0 不会有任何性能上的开销，反倒是得益于 HTTP2.0 的多路复用等技术，后续可以节约大量时间。
- 如果把 HTTPS2.0 当做目标，那么 HTTPS 的性能损耗就更小了，远远比不上它带来的安全性提升。

# HTTP2




# 什么是 HTTP2.0？

HTTP2.0，可以说是SPDY的升级版（其实原本也是基于SPDY设计的），但是，HTTP2.0 跟 SPDY 仍有不同的地方，如下：

- HTTP2.0 支持明文 HTTP 传输，而 SPDY 强制使用 HTTPS。
- HTTP2.0 消息头的压缩算法采用 HPACK，而非 SPDY 采用的 DEFLATE。

# HTTP2.0 和 HTTP1.X 相比的新特性?

- 1、新的二进制格式（Binary Format）

  > HTTP1.x 的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认 0 和 1 的组合。基于这种考虑 HTTP2.0 的协议解析决定采用二进制格式，实现方便且健壮。

- 同 SPDY 对 HTTP1.1 的改进。
  - 2、降低延迟
  - 3、多路复用（MultiPlexing）
  - 4、header 压缩
  - 5、服务端推送（server push）

# Nginx 怎么做 HTTP2.0 的升级改造?

- 1、虽然 HTTP2.0 其实可以支持非 HTTPS 的，但是现在主流的浏览器像 Chrome，Firefox 表示还是只支持基于 TLS 部署的 HTTP2.0协议，所以要想升级成 HTTP2.0 还是先升级 HTTPS 为好。

- 2、当你的网站已经升级 HTTPS 之后，那么升级 HTTP2.0 就简单很多，如果你使用 NGINX ，只要在配置文件中启动相应的协议就可以了，可以参考 NGINX白皮书，NGINX配置HTTP2.0官方指南 。
- 3、使用了 HTTP2.0 那么，原本的 HTTP1.x 怎么办？这个问题其实不用担心，HTTP2.0 完全兼容 HTTP1.x 的语义，对于不支持 HTTP2.0 的浏览器，NGINX 会自动向下兼容的。

在我们内部的微服务 API 接口，也可以做 HTTP2 的改造，可以参考如下文章：

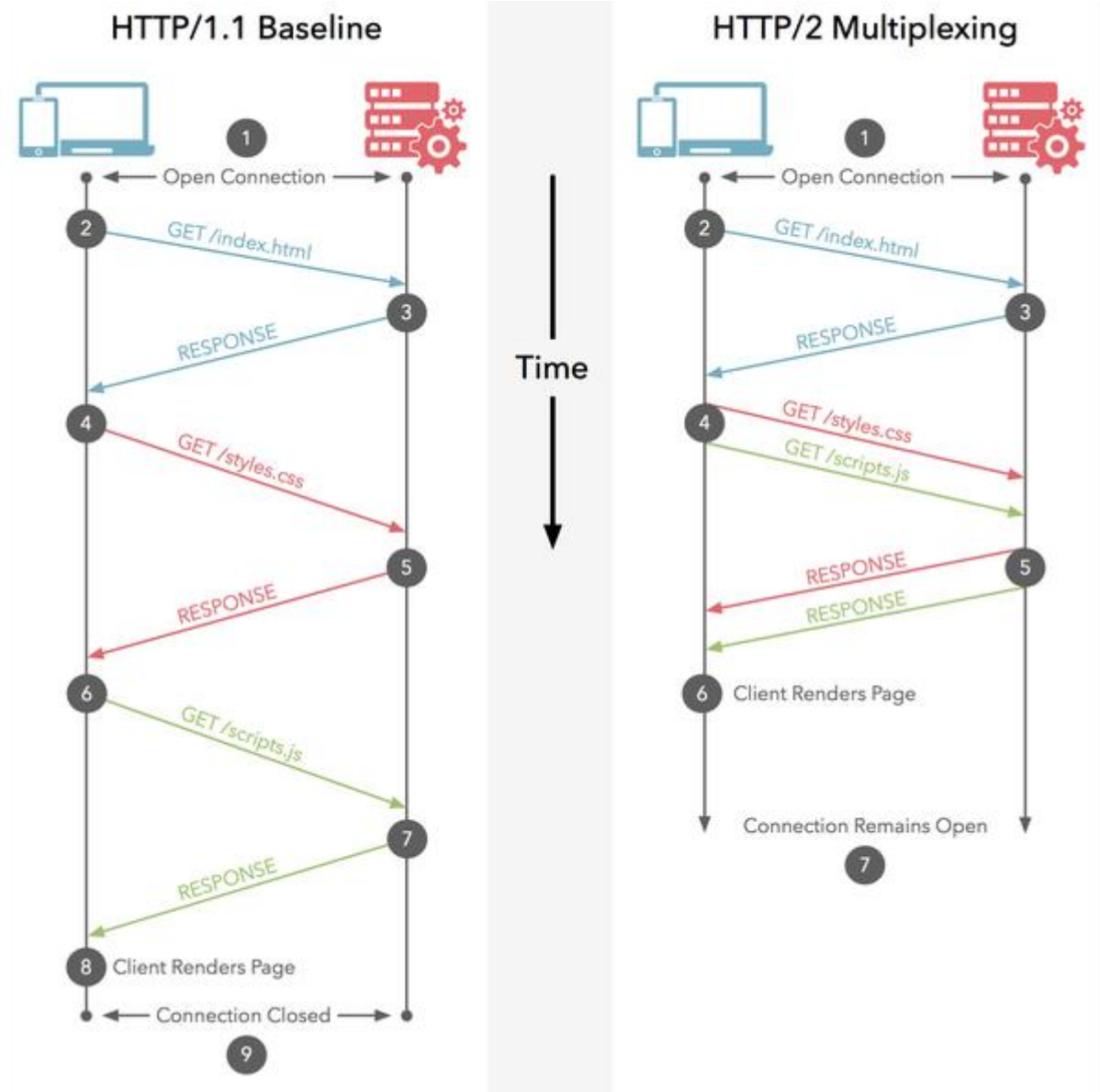
> ：选读，作为知识的扩充

- 《Spring Cloud 使用 HTTP2》
- 《再说 SpringBoot2.0 与 HTTP/2》

# HTTP2.0 的多路复用和 HTTP1.X 中的长连接复用有什么区别?

- HTTP/1.0：一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接。
- HTTP/1.1：Pipeling 解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会。一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞。
- HTTP/2：多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行。

如下图所示：




**HTTP2.0 多路复用有多好?**

> ：最近重新看了 TCP ，发现再看这个问题，真的是有趣!

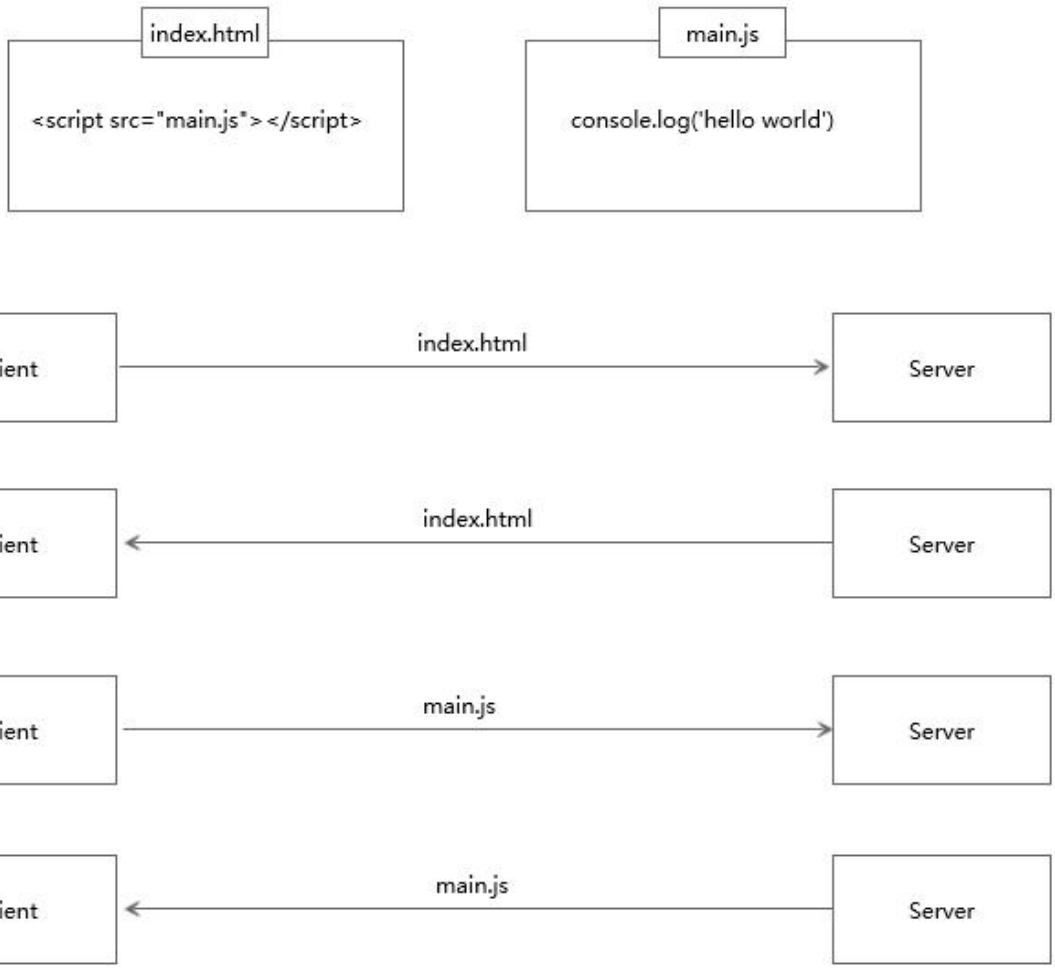
HTTP 性能优化的关键并不在于高带宽，而是低延迟。TCP 连接会随着时间进行自我「调谐」，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调谐则被称为 TCP 慢启动。由于这种原因，让原本就具有突发性和短时性的 HTTP 连接变的十分低效。

HTTP/2 通过让所有数据流共用同一个连接，可以更有效地使用 TCP 连接，让高带宽也能真正的服务于 HTTP 的性能提升。

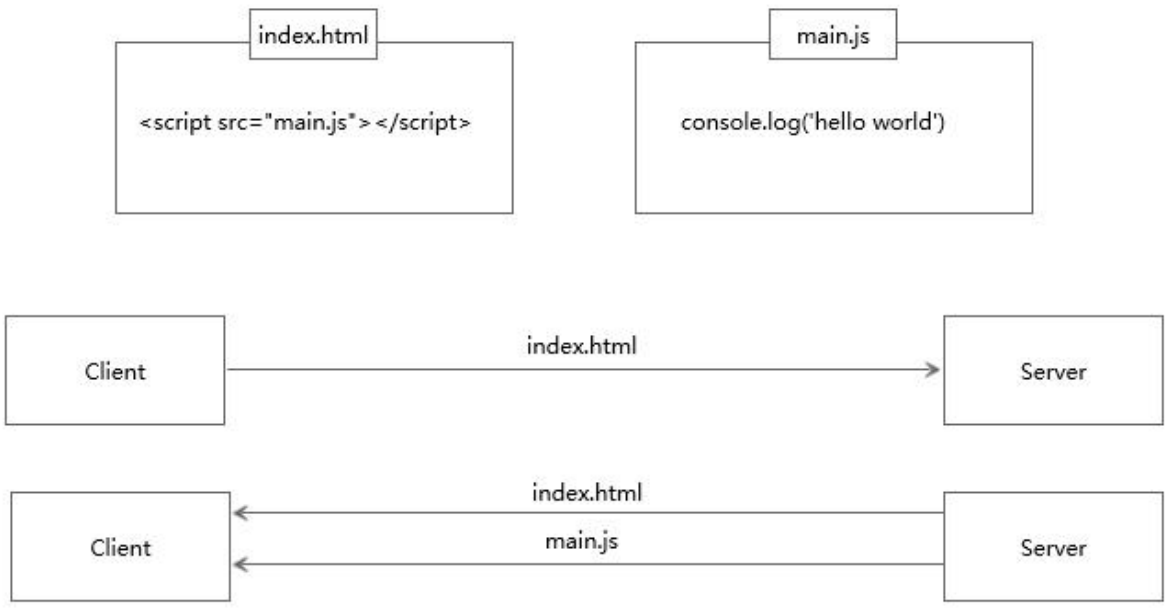
# 服务器推送到底是什么?

服务端推送能把客户端所需要的资源伴随着 `index.html` 一起发送到客户端，省去了客户端重复请求的步骤。正因为没有发起请求，建立连接等操作，所以静态资源通过服务端推送的方式可以极大地提升速度。具体如下:

- 普通的客户端请求过程：




- 服务端推送的过程：




# 为什么需要头部(header)压缩?

假定一个页面有 100 个资源需要加载（这个数量对于今天的 Web 而言还是挺保守的），而每一次请求都有 1kb 的消息头（这同样也并不少见，因为 Cookie 和引用等东西的存在），则至少需要多消耗 100kb 来获取这些消息头。HTTP2.0 可以维护一个字典，差量更新 HTTP 头部，大大降低因头部传输产生的流量。

具体参考：《HTTP/2 头部压缩技术介绍》文章。

- 维护一份相同的静态字典（Static Table），包含常见的头部名称，以及特别常见的头部名称与值的组合。
- 维护一份相同的动态字典（Dynamic Table），可以动态地添加内容。
- 支持基于静态哈夫曼码表的哈夫曼编码（Huffman Coding）。

# 彩蛋

感觉，大学丢掉的网络知识，又一次回到我的脑子中了。好开心，又可以遗忘一轮啦，哈哈哈哈。

参考与推荐如下文章：

- 《Https 单向认证和双向认证》
- 《【网络协议】ping 的工作原理》
- 《HTTP1.0、HTTP1.1 和 HTTP2.0 的区别》
- 《计算机网络常见面试题》
- 《总结的网络面试题》
- 《面试/笔试第一弹 —— 计算机网络面试问题集锦》
- 《通俗大白话来理解 TCP 协议的三次握手和四次挥手》
- 《TCP 数据的传输过程》
- 《搞定计算机网络面试，看这篇就够了（补充版）》