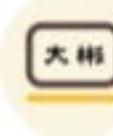


# 计算机网络八股文（2022最新整理）

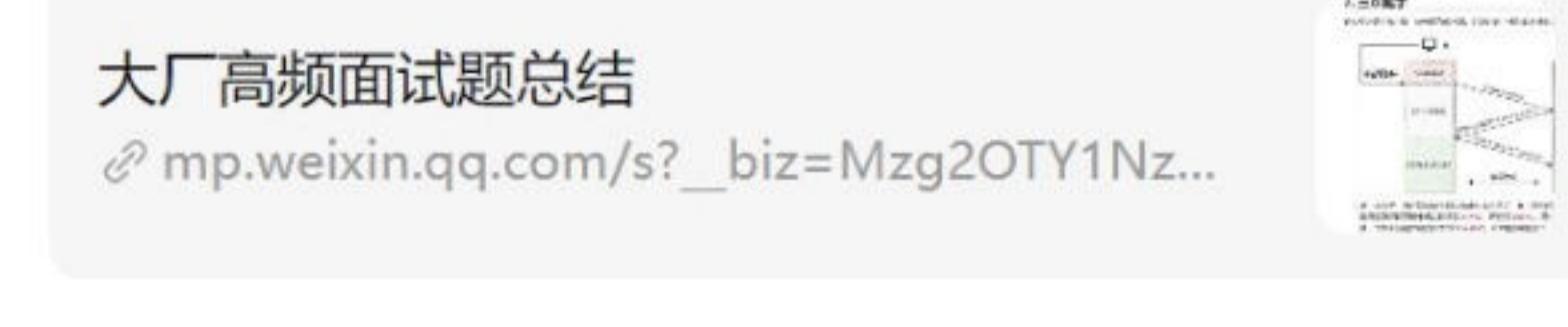


程序员大彬

已关注

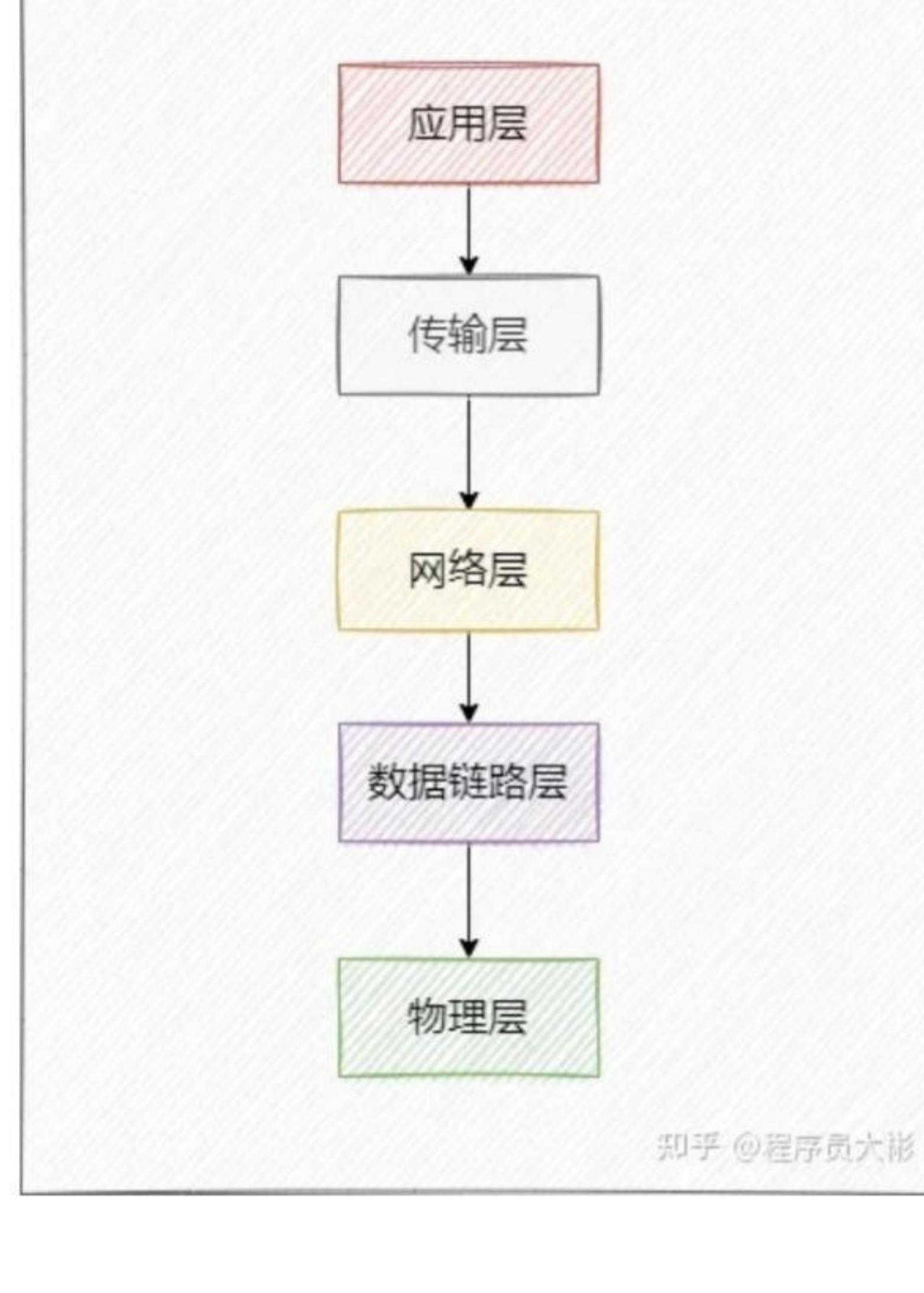
126 人赞同了该文章

给大家分享一份大彬精心整理的**大厂高频面试题PDF**，小伙伴靠着这份手册拿过字节offer，需要的小伙伴可以自行下载：



## 网络分层结构

计算机网络体系大致分为三种，OSI七层模型、TCP/IP四层模型和五层模型。一般面试的时候考察比较多的是五层模型。



**五层模型**：应用层、传输层、网络层、数据链路层、物理层。

- **应用层**：为应用程序提供交互服务。在互联网中的应用层协议很多，如域名系统DNS、HTTP协议、SMTP协议等。
- **传输层**：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**：选择合适的路由和交换结点，确保数据及时传送。主要包括IP协议。
- **数据链路层**：在两个相邻节点之间传送数据时，数据链路层将网络层交下来的IP数据报组装成帧，在两个相邻节点间的链路上发送帧。
- **物理层**：实现相邻节点间比特流的透明传输，尽可能屏蔽传输介质和物理设备的差异。

**ISO七层模型**是国际标准化组织（International Organization for Standardization）制定的一个用于计算机或通信系统间互联的标准体系。

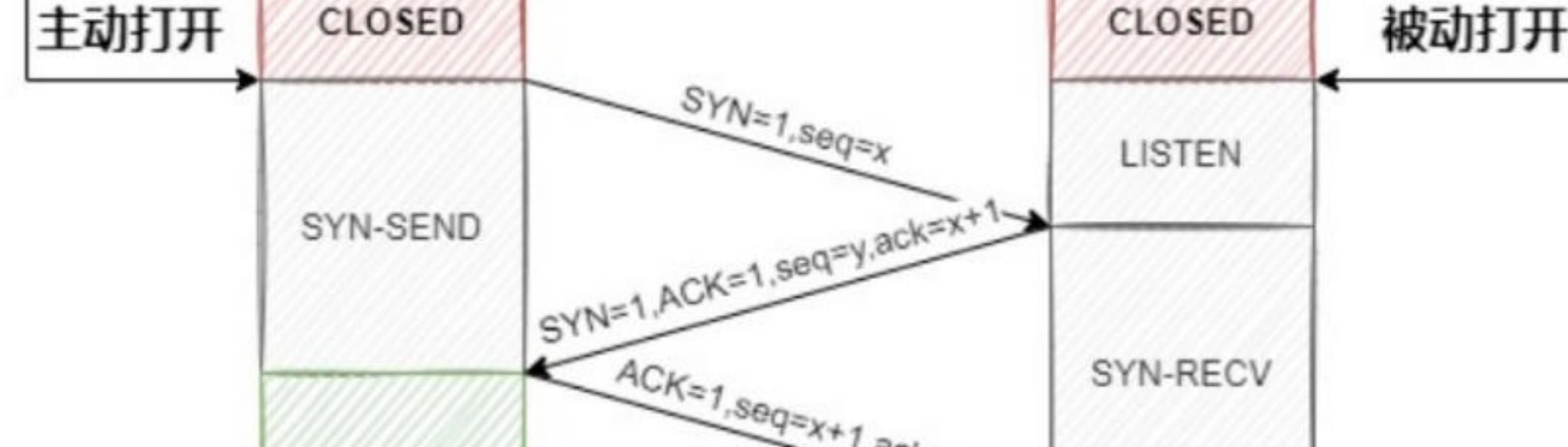
- 应用层：网络服务与最终用户的一个接口，常见的协议有：**HTTP FTP SMTP SNMP DNS**。
- 表示层：数据的表示、安全、压缩。确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。
- 会话层：建立、管理、终止会话，对应主机进程，指本地主机与远程主机正在进行的会话。
- 传输层：定义传输数据的协议端口号，以及流控和差错校验，协议有**TCP UDP**。
- 网络层：进行逻辑地址寻址，实现不同网络之间的路径选择，协议有**ICMP IGMP IP**等。
- 数据链路层：在物理层提供比特流服务的基础上，建立相邻结点之间的数据链路。
- 物理层：建立、维护、断开物理连接。

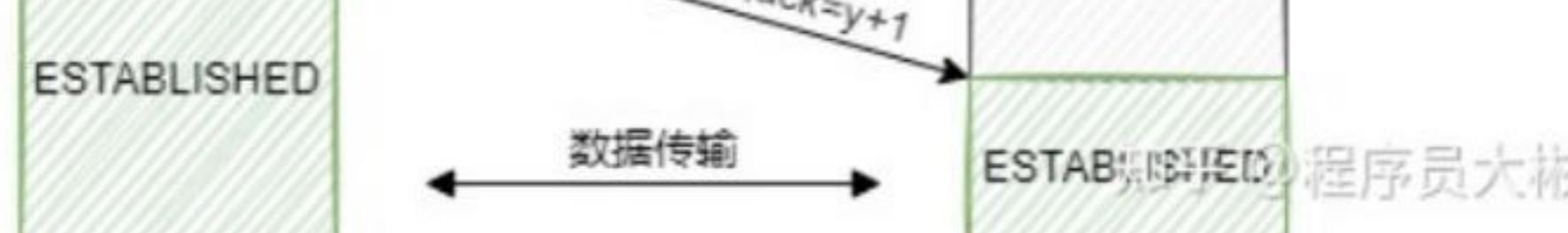
## TCP/IP 四层模型

- 应用层：对应于OSI参考模型的（应用层、表示层、会话层）。
- 传输层：对应OSI的传输层，为应用层实体提供端到端的通信功能，保证了数据包的顺序传送及数据的完整性。
- 网际层：对应于OSI参考模型的网络层，主要解决主机到主机的通信问题。
- 网络接口层：与OSI参考模型的数据链路层、物理层对应。

## 三次握手

假设发送端为客户端，接收端为服务端。开始时客户端和服务端的状态都是 **CLOSED**。





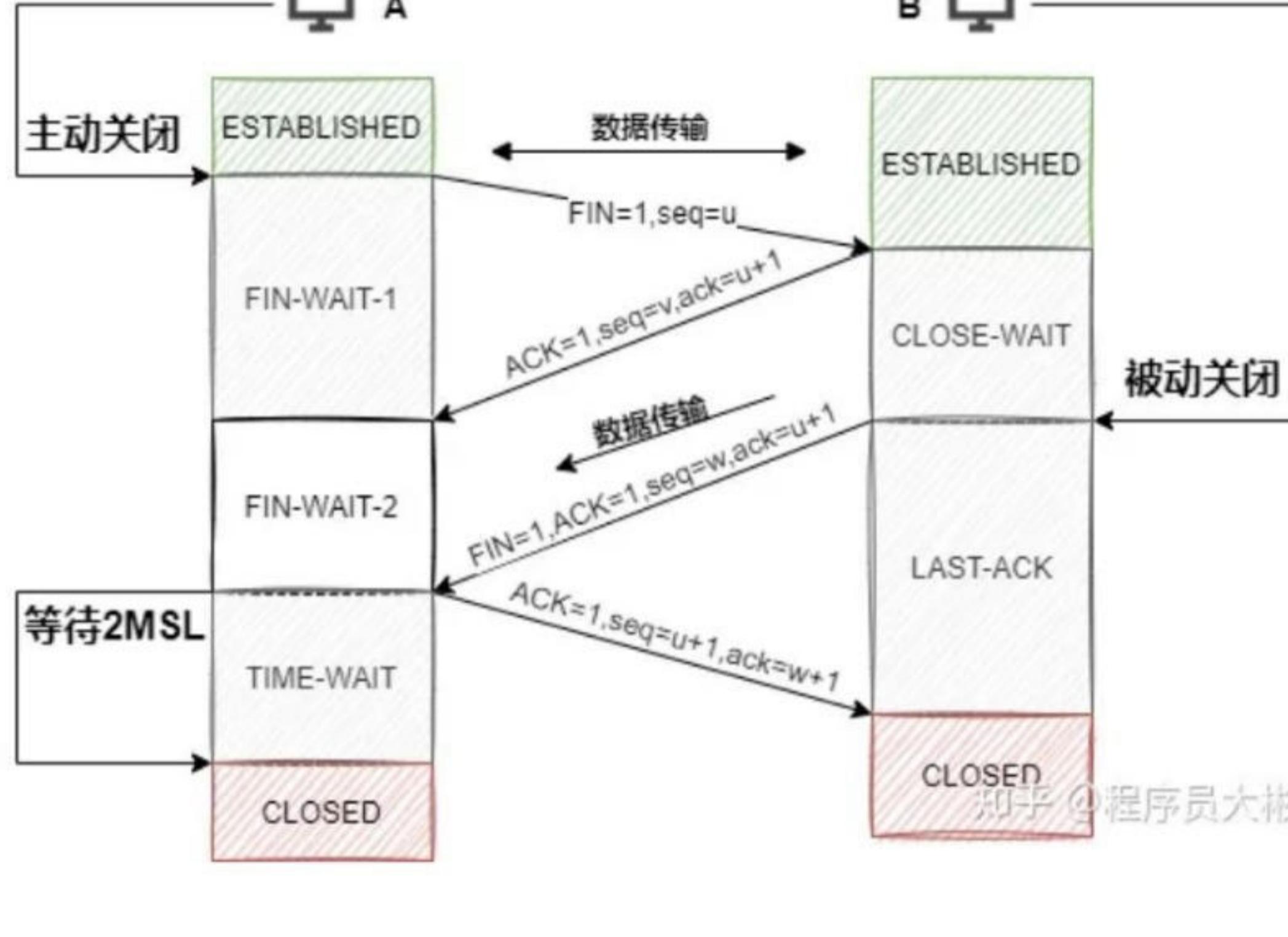
- 第一次握手：客户端向服务端发起建立连接请求，客户端会随机生成一个起始序列号x，客户端向服务端发送的报文段中包含标志位  $SYN=1$ ，序列号  $seq=x$ 。第一次握手前客户端的状态为  $CLOSE$ ，第一次握手后客户端的状态为  $SYN-SENT$ 。此时服务端的状态为  $LISTEN$ 。
- 第二次握手：服务端在收到客户端发来的报文后，会随机生成一个服务端的起始序列号y，然后给客户端回复一段报文，其中包括标志位  $SYN=1$ ， $ACK=1$ ，序列号  $seq=y$ ，确认号  $ack=x+1$ 。第二次握手前服务端的状态为  $LISTEN$ ，第二次握手后服务端的状态为  $SYN-RCVD$ ，此时客户端的状态为  $SYN-SENT$ 。（其中  $SYN=1$  表示要和客户端建立一个连接， $ACK=1$  表示确认序号有效）
- 第三次握手：客户端收到服务端发来的报文后，会再向服务端发送报文，其中包含标志位  $ACK=1$ ，序列号  $seq=x+1$ ，确认号  $ack=y+1$ 。第三次握手前客户端的状态为  $SYN-SENT$ ，第三次握手后客户端和服务端的状态都为  $ESTABLISHED$ 。**此时连接建立完成。**

## 两次握手可以吗？

第三次握手主要为了**防止已失效的连接请求报文段**突然又传输到了服务端，导致产生问题。

- 比如客户端A发出连接请求，可能因为网络阻塞原因，A没有收到确认报文，于是A再重传一次连接请求。
- 连接成功，等待数据传输完毕后，就释放了连接。
- 然后A发出的第一个连接请求等到连接释放以后的某个时间才到达服务端B，此时B误认为A又发出一次新的连接请求，于是就向A发出确认报文段。
- 如果不采用三次握手，只要B发出确认，就建立新的连接了，**此时A不会响应B的确认且不发送数据，则B一直等待A发送数据，浪费资源。**

## 四次挥手



- A的应用进程先向其TCP发出连接释放报文段（ $FIN=1$ ,  $seq=u$ ），并停止再发送数据，主动关闭TCP连接，进入  $FIN-WAIT-1$ （终止等待1）状态，等待B的确认。
- B收到连接释放报文段后即发出确认报文段（ $ACK=1$ ,  $ack=u+1$ ,  $seq=v$ ），B进入  $CLOSE-WAIT$ （关闭等待）状态，此时的TCP处于半关闭状态，A到B的连接释放。
- A收到B的确认后，进入  $FIN-WAIT-2$ （终止等待2）状态，等待B发出的连接释放报文段。
- B发送完数据，就会发出连接释放报文段（ $FIN=1$ ,  $ACK=1$ ,  $seq=w$ ,  $ack=u+1$ ），B进入  $LAST-ACK$ （最后确认）状态，等待A的确认。
- A收到B的连接释放报文段后，对此发出确认报文段（ $ACK=1$ ,  $seq=u+1$ ,  $ack=w+1$ ），A进入  $TIME-WAIT$ （时间等待）状态。此时TCP未释放掉，需要经过时间等待计时器设置的时间  $2MSL$ （最大报文段生存时间）后，A才进入  $CLOSED$  状态。B收到A发出的确认报文段后关闭连接，若没收到A发出的确认报文段，B就会重传连接释放报文段。

## 第四次挥手为什么要等待2MSL？

- 保证A发送的最后一个ACK报文段能够到达B。**这个 ACK 报文段有可能丢失，B收不到这个确认报文，就会超时重传连接释放报文段，然后A可以在  $2MSL$  时间内收到这个重传的连接释放报文段，接着A重传一次确认，重新启动 $2MSL$ 计时器，最后A和B都进入到  $CLOSED$  状态，若A在  $TIME-WAIT$  状态不等待一段时间，而是发送完ACK报文段后立即释放连接，则无法收到B重传的连接释放报文段，所以不会再发送一次确认报文段，B就无法正常进入到  $CLOSED$  状态。
- 防止已失效的连接请求报文段出现在本连接中。**A在发送完最后一个 ACK 报文段后，再经过  $2MSL$ ，就可以使这个连接所产生的所有报文段都从网络中消失，使下一个新的连接中不会出现旧的连接请求报文段。

## 为什么是四次挥手？

因为当Server端收到Client端的  $SYN$  连接请求报文后，可以直接发送  $SYN+ACK$  报文。**但是在关闭连接时，当Server端收到Client端发出的连接释放报文时，很可能并不会立即关闭SOCKET**，所以Server端先回复一个  $ACK$  报文，告诉Client端我收到你的连接释放报文了。只有等到Server端所有的报文都发送完了，这时Server端才能发送连接释放报文，之后两边才会真正的断开连接。故需要四次挥手。

## TCP有哪些特点？

- TCP是**面向连接**的运输层协议。

- 点对点**，每一条TCP连接只能有两个端点。

- TCP提供**可靠交付**的服务。

- TCP提供**全双工通信**。

- 面向字节流。

另外送大家一本谷歌大佬撰写的算法手册，整整 300 道 LeetCode 题目，并且都是最优解，非常强！这本手册帮助不少朋友加入大厂，大家加油！



- **16位端口号**: 源端口号，主机该报文段是来自哪里；目标端口号，要传给哪个上层协议或应用程序
- **32位序号**: 一次TCP通信（从TCP连接建立到断开）过程中某一个传输方向上的字节流的每个字节的编号。
- **32位确认号**: 用作对另一方发送的tcp报文段的响应。其值是收到的TCP报文段的序号值加1。
- **4位头部长度**: 表示tcp头部有多少个32bit字（4字节）。因为4位最大能标识15，所以TCP头部最长是60字节。
- **6位标志位**: URG(紧急指针是否有效), ACK (表示确认号是否有效) , PSH (缓冲区尚未填满) , RST (表示要求对方重新建立连接) , SYN (建立连接消息标志接) , FIN (表示告知对方本端要关闭连接了)
- **16位窗口大小**: 是TCP流量控制的一个手段。这里说的窗口，指的是接收通告窗口。它告诉对方本端的TCP接收缓冲区还能容纳多少字节的数据，这样对方就可以控制发送数据的速度。
- **16位校验和**: 由发送端填充，接收端对TCP报文段执行CRC算法以检验TCP报文段在传输过程中是否损坏。注意，这个校验不仅包括TCP头部，也包括数据部分。这也是TCP可靠传输的一个重要保障。
- **16位紧急指针**: 一个正的偏移量。它和序号字段的值相加表示最后一个紧急数据的下一字节的序号。因此，确切地说，这个字段是紧急指针相对当前序号的偏移，不妨称之为紧急偏移。TCP的紧急指针是发送端向接收端发送紧急数据的方法。

## TCP和UDP的区别？

1. **TCP面向连接**； UDP是无连接的，即发送数据之前不需要建立连接。
2. **TCP提供可靠的服务**； UDP不保证可靠交付。
3. **TCP面向字节流**，把数据看成一连串无结构的字节流； UDP是面向报文的。
4. **TCP有拥塞控制**； 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，如实时视频会议等）。
5. 每一条TCP连接只能是**点到点的**； UDP支持一对一、一对多、多对一和多对多的通信方式。
6. TCP首部开销20字节； UDP的首部开销小，只有8个字节。

## TCP 和 UDP 分别对应的常见应用层协议有哪些？

**基于TCP的应用层协议有：HTTP、FTP、SMTP、TELNET、SSH**

- **HTTP**: HyperText Transfer Protocol (超文本传输协议) , 默认端口80
- **FTP**: File Transfer Protocol (文件传输协议), 默认端口20用于传输数据, 21用于传输控制信息)
- **SMTP**: Simple Mail Transfer Protocol (简单邮件传输协议) ,默认端口25
- **TELNET**: Teletype over the Network (网络电传), 默认端口23
- **SSH**: Secure Shell (安全外壳协议) , 默认端口 22

**基于UDP的应用层协议：DNS、TFTP、SNMP**

- **DNS** : Domain Name Service (域名服务),默认端口 53
- **TFTP**: Trivial File Transfer Protocol (简单文件传输协议), 默认端口69
- **SNMP**: Simple Network Management Protocol (简单网络管理协议) , 通过UDP端口161接收，只有Trap信息采用UDP端口162。

## TCP的粘包和拆包

TCP是面向流，没有界限的一串数据。TCP底层并不了解上层业务数据的具体含义，它会根据TCP缓冲区的实际情况进行包的划分，所以在业务上认为，**一个完整的包可能会被TCP拆分成多个包进行发送，也有可能把多个小的包封装成一个大的数据包发送**，这就是所谓的TCP粘包和拆包问题。

### 为什么会产生粘包和拆包呢？

- 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；
- 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。即TCP报文长度-TCP头部长度>MSS。

解决方案

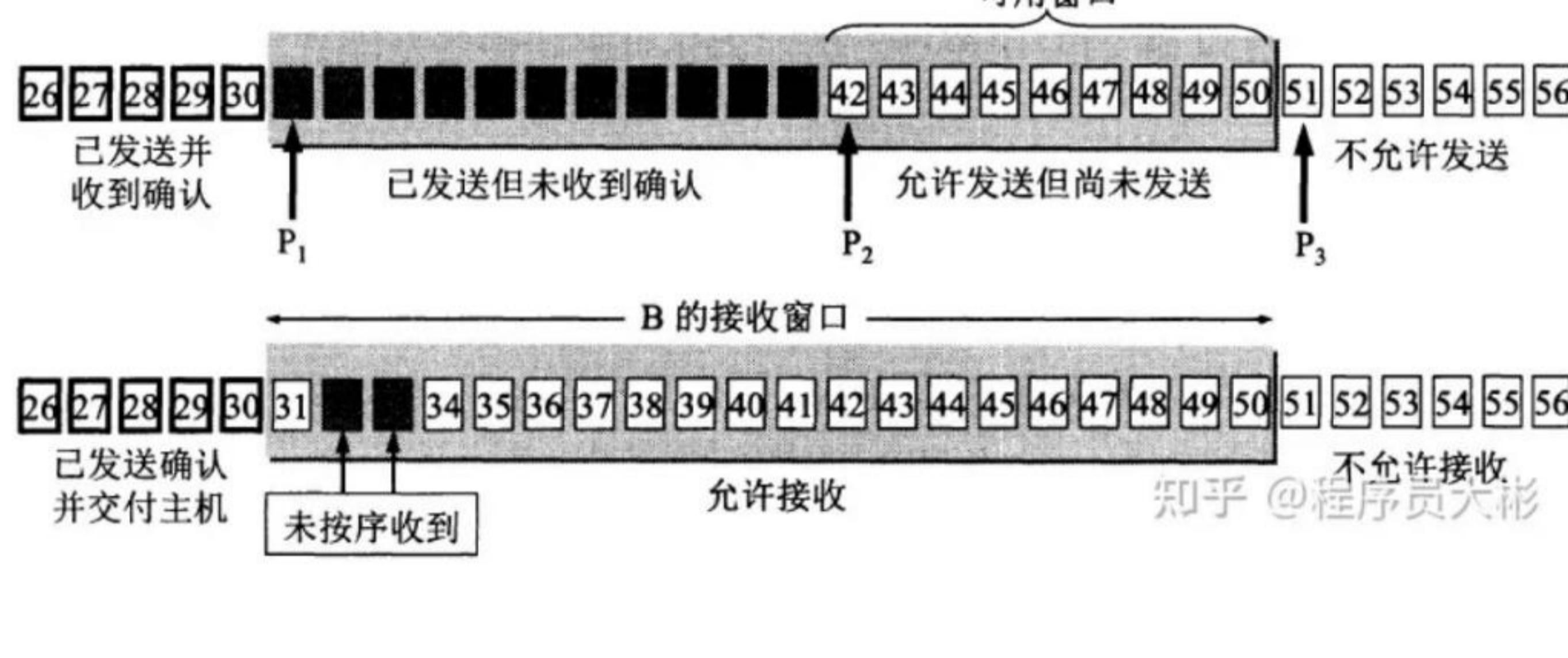
- 发送端将每个数据包封装为固定长度
  - 在数据尾部增加特殊字符进行分割
  - 将数据分为两部分，一部分是头部，一部分是内容体；其中头部结构大小固定，且有一个字段声明内容体的大小。

**说说TCP是如何确保可靠性的呢？**

- 首先，TCP的连接是基于**三次握手**，而断开则是基于**四次挥手**。确保连接和断开的可靠性。
  - 其次，TCP的可靠性，还体现在**有状态**；TCP会记录哪些数据发送了，哪些数据被接收了，哪些没有被接受，并且保证数据包按序到达，保证数据传输不出差错。
  - 再次，TCP的可靠性，还体现在**可控制**。它有数据包校验、ACK应答、**超时重传(发送方)**、失序数据重传（接收方）、丢弃重复数据、流量控制（滑动窗口）和拥塞控制等机制。

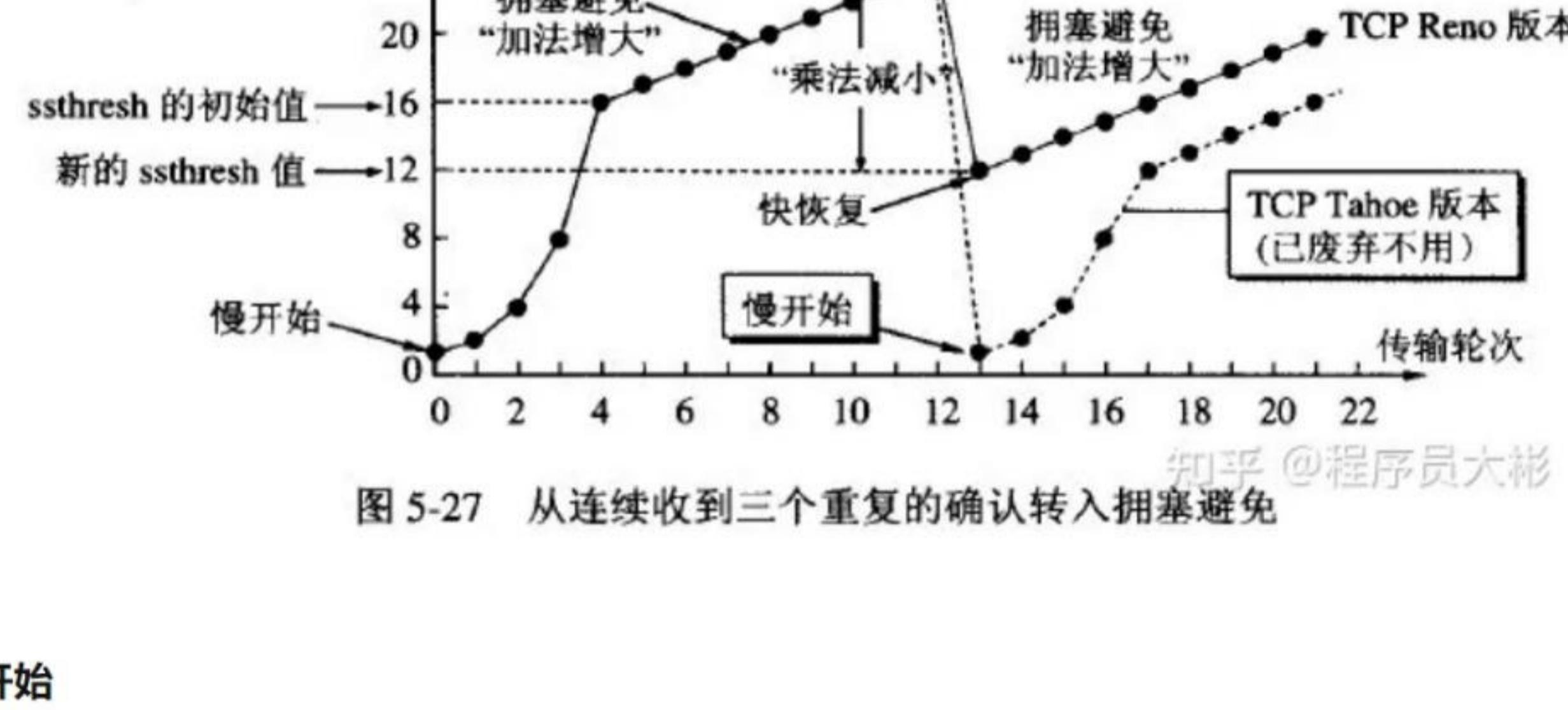
www.english-test.net

TCP会话的双方都各自维护一个发送窗口和一个接收窗口。接收窗口大小取决于应用、系统、硬件的限制。发送窗口则取决于对端通告的接收窗口。接收方发送的确认报文中的window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。将接收方的确认报文window字段设置为0，则发送方不能发送数据。



发送数据，而不会导致接收端处理不过来。接收窗口的大小是约等于发送窗口的大小。

防止过多的数据注入到网络



把拥塞

当  $cwnd < ssthresh$  时，使用慢开始算法。  
当  $cwnd > ssthresh$  时，停止使用慢开始算法而改用拥塞避免算法。

当 `swpd = setbreak` 时，既可使用慢开始。

## 拥堵避免

让拥塞窗口 $cwnd$ 缓慢地增大，每经过一个往返时间RTT就把发送方的拥塞窗口

加倍。这

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（其依据就是没有收到确认），就要把慢开始门限 $ssthresh$ 设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口 $cwnd$ 重新设置为1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网上的数据报文，从而减少拥塞的路由器对拥塞的处理时间。

快重传

有时个别报文段会在网络中丢失，但实际上网络并未发生拥塞。如果发送方迟迟收不到确认，就会产生超时，就会误认为网络发生了拥塞。这就导致发送方错误地启动慢开始，把拥塞窗口 $cwnd$ 又设置为1，因而降低了传输效率。

快重传算法可以避免这个问题。快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认，使发送方及早知道有报文段没有到达对方。

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，而不必继续等待重传计时器到期。由于发送方尽早重传未被确认的报文段，因此采用快重传后可以使整个网络吞吐量提高约20%。

### 快恢复

当发送方连续收到三个重复确认，就会把慢开始门限ssthresh减半，接着把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法，使拥塞窗口缓慢地线性增大。

在采用快恢复算法时，慢开始算法只是在TCP连接建立时和网络出现超时时才使用。采用这样的拥塞控制方法使得TCP的性能有明显的改进。

## HTTP协议的特点？

1. HTTP允许传输**任意类型**的数据。传输的类型由Content-Type加以标记。
2. **无状态**。对于客户端每次发送的请求，服务器都认为是一个新的请求，上一次会话和下一次会话之间没有联系。
3. 支持**客户端/服务器模式**。

## HTTP报文格式

HTTP请求由**请求行、请求头部、空行和请求体**四个部分组成。

- **请求行**：包括请求方法，访问的资源URL，使用的HTTP版本。`GET` 和 `POST` 是最常见的HTTP方法，除此以外还包括 `DELETE`、`HEAD`、`OPTIONS`、`PUT`、`TRACE`。
- **请求头**：格式为“属性名:属性值”，服务端根据请求头获取客户端的信息，主要有 `cookie`、`host`、`connection`、`accept-language`、`accept-encoding`、`user-agent`。
- **请求体**：用户的请求数据如用户名，密码等。

### 请求报文示例：

```
POST /xxx HTTP/1.1 请求行
Accept:image/gif,image/jpeg, 请求头部
Accept-Language:zh-cn
Connection:Keep-Alive
Host:localhost
User-Agent:Mozilla/4.0(compatible;MSIE5.01;Windows NT5.0)
Accept-Encoding:gzip,deflate

username=dabin 请求体
```

HTTP响应也由四个部分组成，分别是：**状态行、响应头、空行和响应体**。

- **状态行**：协议版本，状态码及状态描述。
- **响应头**：响应头字段主要有 `connection`、`content-type`、`content-encoding`、`content-length`、`set-cookie`、`Last-Modified`、`Cache-Control`、`Expires`。
- **响应体**：服务器返回给客户端的内容。

### 响应报文示例：

```
HTTP/1.1 200 OK
Server:Apache Tomcat/5.0.12
Date:Mon, 6 Oct 2003 13:23:42 GMT
Content-Length:112

<html>
    <body>响应体</body>
</html>
```

## HTTP状态码有哪些？

1xx	服务器收到请求，需要请求者继续执行操作
2xx	请求正常处理完毕
3xx	重定向，需要进一步操作已完成请求
4xx	客户端错误，服务器无法处理请求
5xx	服务器处理请求出错

## HTTP 协议包括哪些请求？

HTTP协议中共定义了八种方法来表示对Request-URI指定的资源的不同操作方式，具体如下：

- GET：向特定的资源发出请求。
- POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的创建和/或已有资源的修改。
- OPTIONS：返回服务器针对特定资源所支持的HTTP请求方法。也可以利用向Web服务器发送'\*'的请求来测试服务器的功能性。
- HEAD：向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以在

不必传输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。

- PUT：向指定资源位置上传其最新内容。
- DELETE：请求服务器删除Request-URI所标识的资源。
- TRACE：回显服务器收到的请求，主要用于测试或诊断。
- 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。

## HTTP状态码301和302的区别？

- 301：（永久性转移）请求的网页已被永久移动到新位置。服务器返回此响应时，会自动将请求者转到新位置。
- 302：（暂时性转移）服务器目前正从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。此代码与响应GET和HEAD请求的301代码类似，会自动将请求者转到不同的位置。

**举个形象的例子：**当一个网站或者网页24—48小时内临时移动到一个新的位置，这时候就要进行302跳转，打个比方说，我有一套房子，但是最近走亲戚去亲戚家住了，过两天我还回来的。而使用301跳转的场景就是之前的网站因为某种原因需要移除掉，然后要到新的地址访问，是永久性的，就比如你的那套房子其实是租的，现在租期到了，你又在另一个地方找到了房子，之前租的房子不住了。

## URI和URL的区别

- URI，全称是Uniform Resource Identifier)，中文翻译是统一资源标志符，主要作用是唯一标识一个资源。
- URL，全称是Uniform Resource Location)，中文翻译是统一资源定位符，主要作用是提供资源的路径。打个经典比喻吧，URI像是身份证，可以唯一标识一个人，而URL更像一个住址，可以通过URL找到这个人。

## POST和GET的区别？

- GET请求参数通过URL传递，POST的参数放在请求体中。
- GET产生一个TCP数据包；POST产生两个TCP数据包。对于GET方式的请求，浏览器会把请求头和请求体一并发送出去；而对于POST，浏览器先发送请求头，服务器响应100 continue，浏览器再发送请求体。
- GET请求会被浏览器主动缓存，而POST不会，除非手动设置。
- GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

## 如何理解HTTP协议是无状态的

当浏览器第一次发送请求给服务器时，服务器响应了；如果同一个浏览器发起第二次请求给服务器时，它还是会响应，但是呢，服务器不知道你就是刚才的那个浏览器。简言之，服务器不会去记住你是谁，所以是无状态协议。

## HTTP长连接和短连接？

HTTP短连接：浏览器和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。

**HTTP1.0默认使用的是短连接。**

HTTP长连接：指的是**复用TCP连接**。多个HTTP请求可以复用同一个TCP连接，这就节省了TCP连接建立和断开的消耗。

**HTTP/1.1起，默认使用长连接。**要使用长连接，客户端和服务器的HTTP首部的Connection都要设置为keep-alive，才能支持长连接。

## HTTP 如何实现长连接？

HTTP分为长连接和短连接，**本质上说的是TCP的长短连接**。TCP连接是一个双向的通道，它是可以保持一段时间不关闭的，因此TCP连接才具有真正的长连接和短连接这一说法哈。

TCP长连接可以复用一个TCP连接，来发起多次的HTTP请求，这样就可以减少资源消耗，比如一次请求HTML，如果是短连接的话，可能还需要请求后续的JS/CSS。

### 如何设置长连接？

通过在头部（请求和响应头）设置**Connection**字段指定为 `keep-alive`，HTTP/1.0协议支持，但是是默认关闭的，从HTTP/1.1以后，连接默认都是长连接。

## HTTP长连接在什么时候会超时？

HTTP一般会有httpd守护进程，里面可以设置**keep-alive timeout**，当tcp连接闲置超过这个时间就会关闭，也可以在HTTP的header里面设置超时时间。

TCP 的**keep-alive**包含三个参数，支持在系统内核的net.ipv4里面设置；当 TCP 连接之后，闲置了**tcp\_keepalive\_time**，则会发送探测包，如果没有收到对方的ACK，那么会每隔**tcp\_keepalive\_intvl**再发一次，直到发送了**tcp\_keepalive\_probes**，就会丢弃该连接。

## HTTP1.1和 HTTP2.0的区别？

HTTP2.0相比HTTP1.1支持的特性：

- **新的二进制格式：**HTTP1.1 基于文本格式传输数据；HTTP2.0采用二进制格式传输数据，解析更高效。

- **多路复用：**在一个连接里，允许同时发送多个请求或响应，并且这些请求或响应能够并行的传输而不被阻塞，避免 HTTP1.1 出现的“队头堵塞”问题。

- **头部压缩**, HTTP1.1的header带有大量信息, 而且每次都要重复发送; HTTP2.0 把header从数据中分离, 并封装成头帧和数据帧, **使用特定算法压缩头帧**, 有效减少头信息大小。并且 **HTTP2.0在客户端和服务器端记录了之前发送的键值对, 对于相同的数据, 不会重复发送**。比如请求a发送了所有的头信息字段, 请求b则**只需要发送差异数据**, 这样可以减少冗余数据, 降低开销。
- **服务端推送**: HTTP2.0允许服务器向客户端推送资源, 无需客户端发送请求到服务器获取。

## HTTPS与HTTP的区别?

1. HTTP是超文本传输协议, 信息是**明文传输**; HTTPS则是具有**安全性的ssl加密传输协议**。
2. HTTP和HTTPS用的端口不一样, HTTP端口是80, HTTPS是443。
3. HTTPS协议**需要到CA机构申请证书**, 一般需要一定的费用。
4. HTTP运行在TCP协议之上; HTTPS运行在SSL协议之上, SSL运行在TCP协议之上。

## 什么是数字证书?

服务端可以向证书颁发机构CA申请证书, 以避免中间人攻击(防止证书被篡改)。证书包含三部分内容: **证书内容、证书签名算法和签名**, 签名是为了验证身份。

```

▼ Certificates (3113 bytes)
  Certificate Length: 1845
  ▼ Certificate: 3082073130820619a003020102021004028c81c25b282959... (id-at-commonName=*.yinxiang.com,id-at-organizationName=*.yinxiang.com)
    ▼ signedCertificate
      version: v3 (2)
      serialNumber: 0x04028c81c25b28295954916ad75193a2
      ▼ signature (sha256WithRSAEncryption) 签名算法
        Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
        > issuer: rdnSequence (0)
        > validity
        > subject: rdnSequence (0)
        > subjectPublicKeyInfo
        > extensions: 10 items
        > algorithmIdentifier (sha256WithRSAEncryption)
        Padding: 0
        encrypted: 8c55a9e56cca74d13881b73b8cf94456357454b4694189fc...
        Certificate Length: 1262

```

知乎 @程序员大彬

服务端把证书传输给浏览器, 浏览器从证书里取公钥。证书可以证明该公钥对应本网站。

### 数字签名的制作过程:

1. CA使用证书签名算法对证书内容进行**hash运算**。
2. 对hash后的值用**CA的私钥加密**, 得到数字签名。

### 浏览器验证过程:

1. 获取证书, 得到证书内容、证书签名算法和数字签名。
2. 用CA机构的公钥**对数字签名解密**(由于是浏览器信任的机构, 所以浏览器会保存它的公钥)。
3. 用证书里的签名算法**对证书内容进行hash运算**。
4. 比较解密后的数字签名和对证书内容做hash运算后得到的哈希值, 相等则表明证书可信。

## HTTPS原理

首先是TCP三次握手, 然后客户端发起一个HTTPS连接建立请求, 客户端先发一个 Client Hello 的包, 然后服务端响应 Server Hello, 接着再给客户端发送它的证书, 然后双方经过密钥交换, 最后使用交换的密钥加解密数据。

1. **协商加密算法**。在 Client Hello 里面客户端会告知服务端自己当前的一些信息, 包括客户端要使用的TLS版本, 支持的加密算法, 要访问的域名, 给服务端生成的一个随机数(Nonce)等。需要提前告知服务器想要访问的域名以便服务器发送相应的域名的证书过来。

```

Transport Layer Security
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303) TLS版本
    Length: 192
    ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 188
      Version: TLS 1.2 (0x0303) 随机数
      > Random: 614943ca266085dc951d0da868dbbad76c5a24beed60a2ba...
      Session ID Length: 0
      Cipher Suites Length: 38
      > Cipher Suites (19 suites) 支持的加密算法
        Compression Methods Length: 1
        > Compression Methods (1 method)
        Extensions Length: 109
      ▼ Extension: server_name (len=35)
        Type: server_name (0)
        Length: 35
        ▼ Server Name Indication extension
          Server Name list length: 33
          Server Name Type: host_name (0)
          Server Name length: 30 访问的域名
          Server Name: smartscreen-prod.microsoft.com
        > Extension: status_request (len=5)
      ▼ Extension: supported_groups (len=8)
        Type: supported_groups (10)
        Length: 8

```

知乎 @程序员大彬

1. 服务端响应 Server Hello, 告诉客户端服务端**选中的加密算法**。

```

  ▼ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 85
    Version: TLS 1.2 (0x0303)
    > Random: 614943cafee35db778aae4d42ea345a3caf993d7cc2c494e...
    Session ID Length: 32
    Session ID: e0020000bd5d61c139c863718952b88aa453e1559d08a820...
    > Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Compression Method: null (0)

```

```
Extensions Length: 13
> Extension: status_request (len=0)
> Extension: extended_master_secret (len=0)
> Extension: renegotiation_info (len=1)
```

知乎 @程序员大彬

- 接着服务端给客户端发来了2个证书。第二个证书是第一个证书的签发机构（CA）的证书。

```
✓ Handshake Protocol: Certificate
  Handshake Type: Certificate (11)
  Length: 4874
  Certificates Length: 4871
  ✓ Certificates (4871 bytes)
    Certificate Length: 3338
    > Certificate: 30820d0630820aeea0030201020213330014e91da0c59c8e...
    Certificate Length: 1527
    > Certificate: 308205f3308204dba003020102021002e79171fb8021e93f...
```

知乎 @程序员大彬

- 客户端使用证书的认证机构CA公开发布的RSA公钥对该证书进行验证，下图表明证书认证成功。

```
✓ Handshake Protocol: Certificate Status
  Handshake Type: Certificate Status (22)
  Length: 1731
  Certificate Status Type: OCSP (1)
  OCSP Response Length: 1727
  ✓ OCSP Response
    responseStatus: successful (0)
    > responseBytes
```

知乎 @程序员大彬

- 验证通过之后，浏览器和服务器通过密钥交换算法产生共享的对称密钥。

```
✓ Handshake Protocol: Server Key Exchange
  Handshake Type: Server Key Exchange (12)
  Length: 361
  ✓ EC Diffie-Hellman Server Params
    Curve Type: named_curve (0x03)
    Named Curve: secp384r1 (0x0018)
    Pubkey Length: 97
    Pubkey: 04bf680ce3855152ab100b55fbaba0816ad4f2c675348a8d...
    > Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
    Signature Length: 256
    Signature: 54f207f1e310111e555bfce3cce7e2f3852b1bd4c724be3b...
```

知乎 @程序员大彬

```
✓ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
  Content Type: Handshake (22)
  Version: TLS 1.2 (0x0303)
  Length: 102
  ✓ Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 98
    ✓ EC Diffie-Hellman Client Params
      Pubkey Length: 97
      Pubkey: 04803cee1ce24c5e1ec87ad630b211322045a417f975c858...
```

知乎 @程序员大彬

- 开始传输数据，使用同一个对称密钥来加解密。

```
✓ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
  Content Type: Application Data (23)
  Version: TLS 1.2 (0x0303)
  Length: 445
  Encrypted Application Data: 000000000000000017d213cd4235f491e74325f6363c714...
```

## DNS 的解析过程？

- 浏览器搜索自己的DNS缓存
- 若没有，则搜索操作系统中的DNS缓存和hosts文件
- 若没有，则操作系统将域名发送至本地域名服务器，本地域名服务器查询自己的DNS缓存，查找成功则返回结果，否则依次向根域名服务器、顶级域名服务器、权限域名服务器发起查询请求，最终返回IP地址给本地域名服务器
- 本地域名服务器将得到的IP地址返回给操作系统，同时自己也将IP地址缓存起来
- 操作系统将IP地址返回给浏览器，同时自己也将IP地址缓存起来
- 浏览器得到域名对应的IP地址

## 浏览器中输入URL返回页面过程？

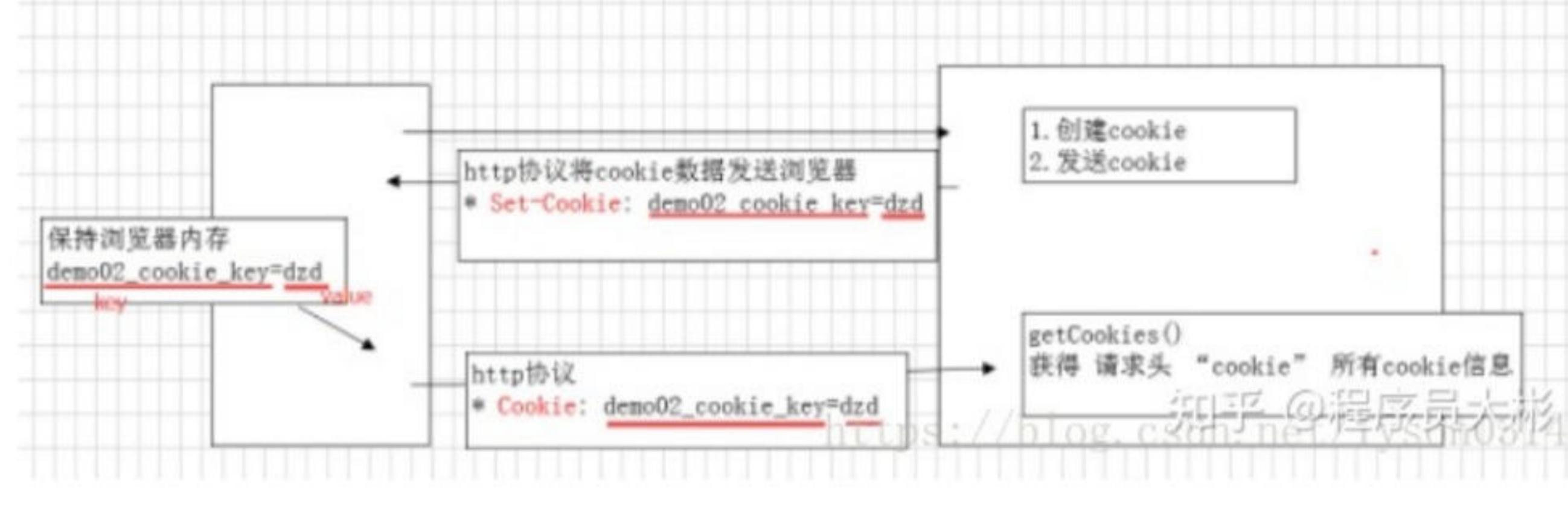
- 解析域名，找到主机IP。
- 浏览器利用IP直接与网站主机通信，**三次握手**，建立TCP连接。浏览器会以一个随机端口向服务端的web程序80端口发起TCP的连接。
- 建立TCP连接后，浏览器向主机发起一个HTTP请求。
- 服务器**响应请求**，返回响应数据。
- 浏览器**解析响应内容**，进行渲染，呈现给用户。



## 什么是cookie和session？

由于HTTP协议是无状态的协议，需要用某种机制来识别具体的用户身份，用来跟踪用户的整个会话。常用的会话跟踪技术是cookie与session。

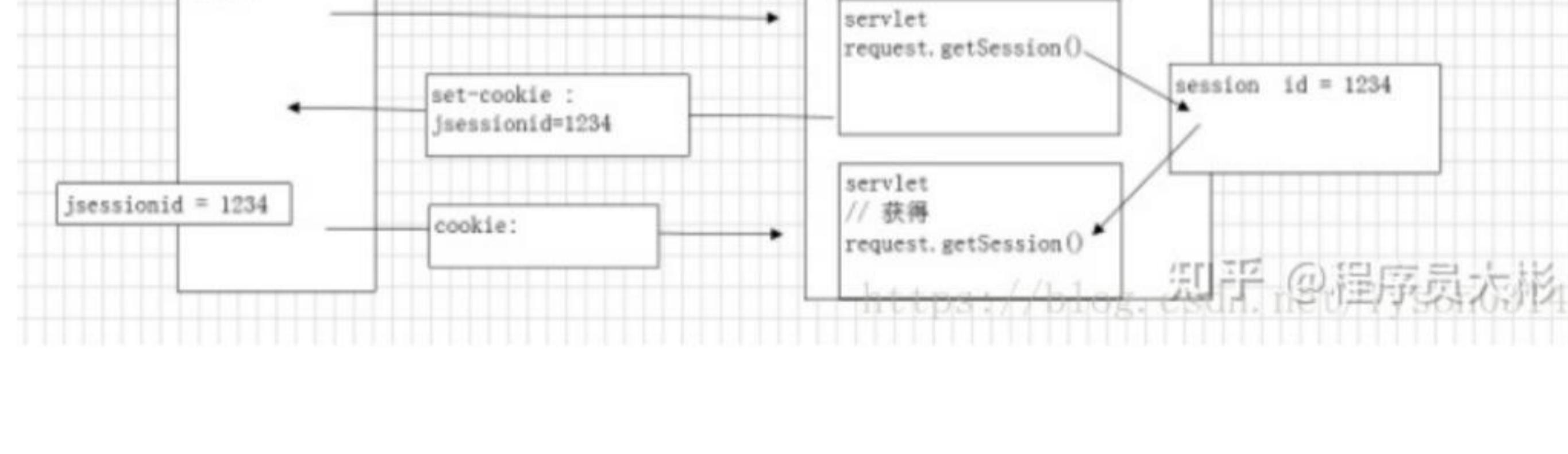
**cookie**就是由服务器发给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。说得更具体一些：当用户使用浏览器访问一个支持cookie的网站的时候，用户会提供包括用户名在内的个人信息并且提交至服务器；接着，服务器在向客户端回传相应的超文本的同时也会发回这些个人信息，当然这些信息并不是存放在HTTP响应体中的，而是存放于HTTP响应头；当客户端浏览器接收到来自服务器的响应之后，浏览器会将这些信息存放在一个统一的位置。自此，客户端再向服务器发送请求的时候，都会把相应的cookie存放在HTTP请求头再次发回至服务器。服务器在接收到来自客户端浏览器的请求之后，就能够通过分析存放于请求头的cookie得到客户端特有的信息，从而动态生成与该客户端相对应的内容。网站的登录界面中“请记住我”这样的选项，就是通过cookie实现的。



### cookie工作流程：

1. servlet创建cookie，保存少量数据，发送给浏览器。
2. 浏览器获得服务器发送的cookie数据，将自动的保存到浏览器端。
3. 下次访问时，浏览器将自动携带cookie数据发送给服务器。

**session原理**：首先浏览器请求服务器访问web站点时，服务器首先会检查这个客户端请求是否已经包含了一个session标识、称为SESSIONID，如果已经包含了一个sessionid则说明以前已经为此客户端创建过session，服务器就按照sessionid把这个session检索出来使用，如果客户端请求不包含session id，则服务器为此客户端创建一个session，并且生成一个与此session相关联的独一无二的sessionid存放到cookie中，这个sessionid将在本次响应中返回到客户端保存，这样在交互的过程中，浏览器端每次请求时，都会带着这个sessionid，服务器根据这个sessionid就可以找得到对应的session。以此来达到共享数据的目的。这里需要注意的是，session不会随着浏览器的关闭而死亡，而是等待超时时间。



### cookie和session的区别？

- **作用范围不同**，Cookie 保存在客户端，Session 保存在服务器端。
- **有效期不同**，Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失效时间较短，客户端关闭或者 Session 超时都会失效。
- **隐私策略不同**，Cookie 存储在客户端，容易被窃取；Session 存储在服务端，安全性相对 Cookie 要好一些。
- **存储大小不同**，单个 Cookie 保存的数据不能超过 4K；对于 Session 来说存储没有上限，但出于对服务器的性能考虑，Session 内不要存放过多的数据，并且需要设置 Session 删除机制。

### 什么是对称加密和非对称加密？

**对称加密**：通信双方使用**相同的密钥**进行加密。特点是加密速度快，但是缺点是密钥泄露会导致密文数据被破解。常见的对称加密有 AES 和 DES 算法。

**非对称加密**：它需要生成两个密钥，**公钥和私钥**。公钥是公开的，任何人都可以获得，而私钥是私人保管的。公钥负责加密，私钥负责解密；或者私钥负责加密，公钥负责解密。这种加密算法**安全性更高**，但是**计算量相比对称加密大很多**，加密和解密都很慢。常见的非对称算法有 RSA 和 DSA。

### 说说 WebSocket与socket的区别

Socket是一套标准，它完成了对TCP/IP的高度封装，屏蔽网络细节，以方便开发者更好地进行网络编程。Socket其实就是等于**IP地址 + 端口 + 协议**。

WebSocket是一个持久化的协议，它是伴随H5而出的协议，用来解决**http不支持持久化连接**的问题。

Socket一个是**网编编程的标准接口**，而WebSocket则是应用层通信协议。

### ARP协议的工作过程？

ARP解决了同一个局域网上的主机和路由器IP和MAC地址的解析。

- 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IP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。

## ICMP协议的功能

ICMP, Internet Control Message Protocol ,Internet控制消息协议。

- ICMP协议是一种面向无连接的协议，用于传输出错报告控制信息。
- 它是一个非常重要的协议，它对于网络安全具有极其重要的意义。它属于网络层协议，主要用于在主机与路由器之间传递控制信息，包括**报告错误、交换受限控制和状态信息等**。
- 当遇到IP数据无法访问目标、IP路由器无法按当前的传输速率转发数据包等情况时，会自动发送ICMP消息。

比如我们日常使用得比较多的**ping**，就是基于ICMP的。

## 什么是DoS、DDoS、DRDoS攻击？

- **DOS**: (Denial of Service),翻译过来就是拒绝服务,一切能引起DOS行为的攻击都被称为DOS攻击。最常见的DoS攻击就有**计算机网络宽带攻击、连通性攻击**。
- **DDoS**: (Distributed Denial of Service),翻译过来是分布式拒绝服务。是指处于不同位置的多个攻击者同时向一个或几个目标发动攻击，或者一个攻击者控制了位于不同位置的多台机器并利用这些机器对受害者同时实施攻击。常见的DDos有**SYN Flood、Ping of Death、ACK Flood、UDP Flood**等。
- **DRDoS**: (Distributed Reflection Denial of Service)，中文是分布式反射拒绝服务，该方式靠的是发送大量带有被害者IP地址的数据包给攻击主机，然后攻击主机对IP地址源做出大量回应，从而形成拒绝服务攻击。

## 什么是CSRF攻击，如何避免

CSRF，跨站请求伪造（英文全称是Cross-site request forgery），是一种挟制用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法。

### 怎么解决CSRF攻击呢？

- 检查Referer字段。

知乎

首发于  
Java基础知识总结

...

写文章

## 什么是XSS攻击？

XSS 跨站脚本攻击 (Cross-Site Scripting)。它指的是恶意攻击者往Web页面里插入恶意html代码当用户浏览该页之时，嵌入其中Web里面的html代码会被执行，从而达到恶意攻击用户的特殊目的。XSS攻击一般分三种类型：**存储型、反射型、DOM型XSS**

赞同 126

分享

## 如何解决XSS攻击问题？

- 对输入进行过滤，过滤标签等，只允许合法值。
- HTML转义
- 对于链接跳转，如 `<a href="xxx"` 等，要校验内容，禁止以script开头的非法链接。
- 限制输入长度

## 说下ping的原理

ping - Packet Internet Groper 目标地址不可达或正在建立连接 D:\> ping 目录  
▲ 赞同 126 ▼ ● 7条评论 分享 喜欢 收藏 申请转载 ...  
Internet Control Message Protocol (因特网控制报文协议) 是为了实现网络中各计算机之间的信息交互而设计的一个协议，了解其有关状态。

一般来说，ping可以用来检测网络通不通。它是基于 ICMP 协议工作的。假设**机器A ping机器B**，工作过程如下：

1. ping通知系统，新建一个固定格式的ICMP请求数据包
2. ICMP协议，将该数据包和目标机器B的IP地址打包，一起转交给IP协议层
3. IP层协议将本机IP地址为源地址，机器B的IP地址为目标地址，加上一些其他的控制信息，构建一个IP数据包
4. 先获取目标机器B的MAC地址。
5. 数据链路层构建一个数据帧，目的地址是IP层传过来的**MAC地址**，源地址是本机的**MAC地址**
6. 机器B收到后，对比目标地址，和自己本机的MAC地址是否一致，符合就处理返回，不符合就丢弃。
7. 根据目的主机返回的ICMP回送回答报文中的时间戳，从而计算出往返时间
8. 最终显示结果有这几项：发送到目的主机的IP地址、发送 & 收到 & 丢失的分组数、往返时间的最小、最大 & 平均值

最后给大家分享**200多本计算机经典书籍PDF电子书**，包括C语言、C++、Java、Python、前端、数据库、操作系统、计算机网络、数据结构和算法、机器学习、编程人生等，感兴趣的小伙伴可以自取：



编辑于 2022-08-10 08:34

计算机网络 Java面试 编程基础



评论千万条，友善第一条

7条评论

默认 最新



ragnarok

是OSI七层模型，不是ISO七层模型

01-25 · IP 属地四川

回复 1



梦夕阳

全称ISO/OSI七层模型

02-16 · IP 属地四川

回复 1



哈喽你白

学到了

2022-07-30 · IP 属地湖北

回复 1



Meiling Wang

看者流泪，闻者伤心😊😊😊

2022-08-29 · IP 属地广东

回复 赞



Meiling Wang › 程序员大彬

难的，看着都难，难死我了🤣🤣

2022-08-29 · IP 属地广东

回复 赞



程序员大彬

作者 › Meiling Wang



2022-08-29 · IP 属地广东

回复 赞

展开其他 1 条回复 >

文章被以下专栏收录



Java基础知识总结

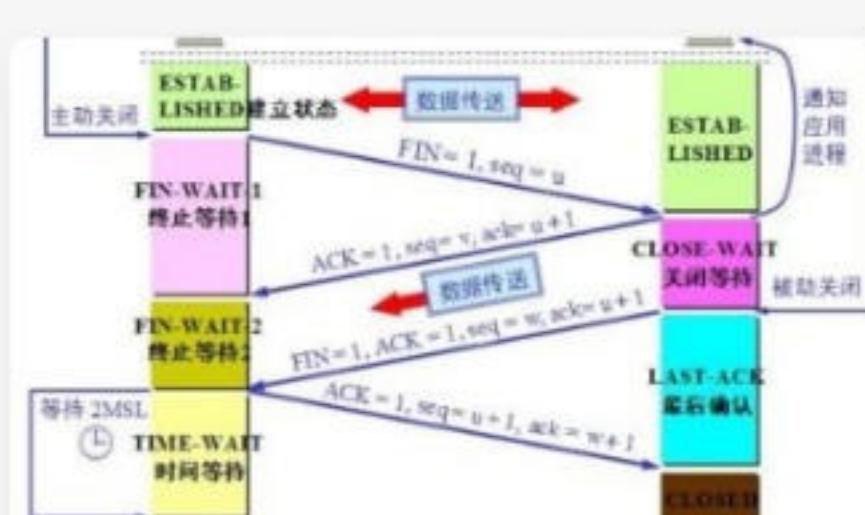
硬核Java知识总结，你值得拥有

推荐阅读

## 2022年最新【计算机网络】八股文汇总（附答案~）

计算机网络是贯穿互联网校招所有技术岗位最高频的面试考点，不论是算法，还是开发，不管是Java、C++，都会问到，这篇文章汇总常见计算机网络八股文，记得先收藏再看！来自面试小抄，完整版...

程序员库森



八股文之计算机网络

千羽

## 计算机网络基础

1、计算机网络的基本概念 1) 计算机网络：利用通信设备和线路把地理上分散的，具有独立功能的多台计算机连接起来，并配以相应的网络软件，从而实现资源共享和信息交换的系统。 2) 计算机网...

xqy 发表于沙漏中的时...

## 计算机网络基础

今天回顾了下计算机网络基础知识，把自己的心得写下来。在上一篇写到，计算机系统的组成包括硬件和软件，硬件比如CPU，网卡，内存，硬盘等等，软件包括操作系统和应用软件，当我们给电脑装...

若谷天豪