计算机网络

1、OSI 的七层模型分别是？各自的功能是什么?

简要概括

- 物理层：底层数据传输，如网线；网卡标准。
- 数据链路层：定义数据的基本格式，如何传输，如何标识；如网卡MAC地址。
- 网络层：定义IP编址，定义路由功能；如不同设备的数据转发。
- 传输层：端到端传输数据的基本功能；如 TCP、UDP。
- 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。
- 表示层：数据格式标识，基本压缩加密功能。
- 应用层：各种应用软件，包括 Web 应用。

说明:

- 在四层，既传输层数据被称作**段**（Segments）；
- 三层网络层数据被称做**包**（Packages）；
- 二层数据链路层时数据被称为**帧**（Frames）；
- 一层物理层时数据被称为**比特流**（Bits）。

总结

- 网络七层模型是一个标准，而非实现。
- 网络四层模型是一个实现的应用模型。
- 网络四层模型由七层模型简化合并而来。

2、说一下一次完整的HTTP请求过程包括哪些内容?

第一种回答

- 建立起客户机和服务器连接。
- 建立连接后，客户机发送一个请求给服务器。
- 服务器收到请求给予响应信息。
- 客户端浏览器将返回的内容解析并呈现，断开连接。

第二种回答

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户。

3、你知道DNS是什么?

官方解释：DNS（Domain Name System，域名系统），因特网上作为**域名和IP地址相互映射**的一个**分布式数据库**，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的IP数串。

通过主机名，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。

通俗的讲，我们更习惯于记住一个网站的名字，比如www.baidu.com,而不是记住它的ip地址，比如：167.23.10.2。

4、DNS的工作原理?

将主机域名转换为ip地址，属于应用层协议，使用UDP传输。（DNS应用层协议，以前有个考官问过）

过程：
总结： 浏览器缓存，系统缓存，路由器缓存，IPS服务器缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存。
一、主机向本地域名服务器的查询一般都是采用递归查询。
二、本地域名服务器向根域名服务器的查询的迭代查询。
1)当用户输入域名时，浏览器先检查自己的缓存中是否 这个域名映射的ip地址，有解析结束。
2）若没命中，则检查操作系统缓存（如Windows的hosts）中有没有解析过的结果，有解析结束。
3）若无命中，则请求本地域名服务器解析（ LDNS）。
4）若LDNS没有命中就直接跳到根域名服务器请求解析。根域名服务器返回给LDNS一个 主域名服务器地址。
5） 此时LDNS再发送请求给上一步返回的gTLD（ 通用顶级域）， 接受请求的gTLD查找并返回这个域名对应的Name Server的地址
6） Name Server根据映射关系表找到目标ip，返回给LDNS
7） LDNS缓存这个域名和对应的ip， 把解析的结果返回给用户，用户根据TTL值缓存到本地系统缓存中，域名解析过程至此结束

5、为什么域名解析用UDP协议?

因为UDP快啊！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。

6、为什么区域传送用TCP协议?

因为TCP协议可靠性好啊!

你要从主DNS上复制内容啊，你用不可靠的UDP？ 因为TCP协议传输的内容大啊，你用最大只能传512字节的UDP协议？万一同步的数据大于512字节，你怎么办？所以用TCP协议比较好!

7、HTTP长连接和短连接的区别

在HTTP/1.0中默认使用短连接。也就是说，客户端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。

而从HTTP/1.1起，默认使用长连接，用以保持连接特性。

8、什么是TCP粘包/拆包？发生的原因?

一个完整的业务可能会被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，这个就是TCP的拆包和粘包问题。

原因

1、应用程序写入数据的字节大小大于套接字发送缓冲区的大小.

2、进行MSS大小的TCP分段。(MSS=TCP报文段长度-TCP首部长度)

3、以太网的payload大于MTU进行IP分片。（ MTU指：一种通信协议的某一层上面所能通过的最大数据包大小。）

解决方案

1、消息定长。

2、在包尾部增加回车或者空格符等特殊字符进行分割3. 将消息分为消息头和消息尾。4. 使用其它复杂的协议，如RTMP协议等。

9、为什么服务器会缓存这一项功能?如何实现的?

原因

- 缓解服务器压力；
- 降低客户端获取资源的延迟：缓存通常位于内存中，读取缓存的速度更快。并且缓存服务器在地理位置上也有可能比源服务器来得近，例如浏览器缓存。

实现方法

- 让代理服务器进行缓存；
- 让客户端浏览器进行缓存。

10、HTTP请求方法你知道多少?

客户端发送的 **请求报文** 第一行为请求行，包含了方法字段。

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头
3	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建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。

序号	方法	描述
5	DELETE	请求服务器删除指定的页面。
6	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。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。
9	PATCH	是对 PUT 方法的补充，用来对已知资源进行局部更新 。

11、GET 和 POST 的区别，你知道哪些?

1. get是获取数据，post是修改数据
2. get把请求的数据放在url上， 以?分割URL和传输数据，参数之间以&相连，所以get不太安全。而post把数据放在HTTP的包体内（requrest body）
3. get提交的数据最大是2k（ 限制实际上取决于浏览器）， post理论上没有限制。
4. GET产生一个TCP数据包，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); POST产生两个TCP数据包，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)。
5. GET请求会被浏览器主动缓存，而POST不会，除非手动设置。
6. 本质区别：GET是幂等的，而POST不是幂等的

> 这里的幂等性：幂等性是指一次和多次请求某一个资源应该具有同样的副作用。简单来说意味着对同一URL的多个请求应该返回同样的结果。

正因为它们有这样的区别，所以不应该且**不能用get请求做数据的增删改这些有副作用的操作。**因为get请求是幂等的，**在网络不好的隧道中会尝试重试。**如果用get请求增数据，会有**重复操作**的风险，而这种重复操作可能会导致副作用（浏览器和操作系统并不知道你会用get请求去做增操作）。

12、一个TCP连接可以对应几个HTTP请求?

如果维持连接，一个 TCP 连接是可以发送多个 HTTP 请求的。

13、一个 TCP 连接中 HTTP 请求发送可以一起发送么（比如一起发三个请求，再三个响应一起接收）？

HTTP/1.1 存在一个问题，单个 TCP 连接在同一时刻只能处理一个请求，意思是说：两个请求的生命周期不能重叠，任意两个 HTTP 请求从开始到结束的时间在同一个 TCP 连接里不能重叠。

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。

那么在 HTTP/1.1 时代，浏览器是如何提高页面加载效率的呢？主要有下面两点：

- 维持和服务器已经建立的 TCP 连接，在同一连接上顺序处理多个请求。
- 和服务器建立多个 TCP 连接。

14、浏览器对同一 Host 建立 TCP 连接到数量有没有限制?

假设我们还处在 HTTP/1.1 时代，那个时候没有多路传输，当浏览器拿到一个有几十张图片的网页该怎么办呢？肯定不能只开一个 TCP 连接顺序下载，那样用户肯定等的很难受，但是如果每个图片都开一个 TCP 连接发 HTTP 请求，那电脑或者服务器都可能受不了，要是有 1000 张图片的话总不能开 1000 个TCP 连接吧，你的电脑同意 NAT 也不一定会同意。

有。Chrome 最多允许对同一个 Host 建立六个 TCP 连接。不同的浏览器有一些区别。

如果图片都是 HTTPS 连接并且在同一个域名下，那么浏览器在 SSL 握手之后会和服务器商量能不能用 HTTP2，如果能的话就使用 Multiplexing 功能在这个连接上进行多路传输。不过也未必会所有挂在这个域名的资源都会使用一个 TCP 连接去获取，但是可以确定的是 Multiplexing 很可能会被用到。

如果发现用不了 HTTP2 呢？或者用不了 HTTPS（现实中的 HTTP2 都是在 HTTPS 上实现的，所以也就是只能使用 HTTP/1.1）。那浏览器就会在一个 HOST 上建立多个 TCP 连接，连接数量的最大限制取决于浏览器设置，这些连接会在空闲的时候被浏览器用来发送新的请求，如果所有的连接都正在发送请求呢？那其他的请求就只能等等了。

15、在浏览器中输入url地址后显示主页的过程?

- 根据域名，进行DNS域名解析；
- 拿到解析的IP地址，建立TCP连接；
- 向IP地址，发送HTTP请求；
- 服务器处理请求；
- 返回响应结果；
- 关闭TCP连接；
- 浏览器解析HTML；
- 浏览器布局渲染；

16、在浏览器地址栏输入一个URL后回车，背后会进行哪些技术步骤?

第一种回答

1、查浏览器缓存，看看有没有已经缓存好的，如果没有

2 、检查本机host文件，

3、调用API，Linux下Scoket函数 gethostbyname

4、向DNS服务器发送DNS请求，查询本地DNS服务器，这其中用的是UDP的协议

6、如果在一个子网内采用ARP地址解析协议进行ARP查询如果不在一个子网那就需要对默认网关进行DNS查询，如果还找不到会一直向上找根DNS服务器，直到最终拿到IP地址（全球好像一共有13台根服务器）

7、这个时候我们就有了服务器的IP地址 以及默认的端口号了，http默认是80 https是 443 端口号，会，首先尝试http然后调用Socket建立TCP连接，

8、经过三次握手成功建立连接后，开始传送数据，如果正是http协议的话，就返回就完事了，

9、如果不是http协议，服务器会返回一个5开头的的重定向消息，告诉我们用的是https，那就是说IP没变，但是端口号从80变成443了，好了，再四次挥手，完事，

10、再来一遍，这次除了上述的端口号从80变成443之外，还会采用SSL的加密技术来保证传输数据的安全性，保证数据传输过程中不被修改或者替换之类的，

11、这次依然是三次握手，沟通好双方使用的认证算法，加密和检验算法，在此过程中也会检验对方的CA安全证书。

12、确认无误后，开始通信，然后服务器就会返回你所要访问的网址的一些数据，在此过程中会将界面进行渲染，牵涉到ajax技术之类的，直到最后我们看到色彩斑斓的网页

第二种回答

浏览器检查域名是否在缓存当中（要查看 Chrome 当中的缓存， 打开 chrome://net-internals/#dns）。

如果缓存中没有，就去调用 `gethostbyname` 库函数（操作系统不同函数也不同）进行查询。

gethostbyname函数在试图进行DNS解析之前首先检查域名是否在本地 `Hosts` 里，`Hosts` 的位置 [不同的操作系统有所不同](https://en.wikipedia.org/wiki/Hosts_(file)#Location_in_the_file_system)

如果gethostbyname没有这个域名的缓存记录，也没有在hosts` 里找到，它将会向 DNS 服务器发送一条 DNS 查询请求。DNS 服务器是由网络通信栈提供的，通常是本地路由器或者 ISP 的缓存 DNS 服务器。

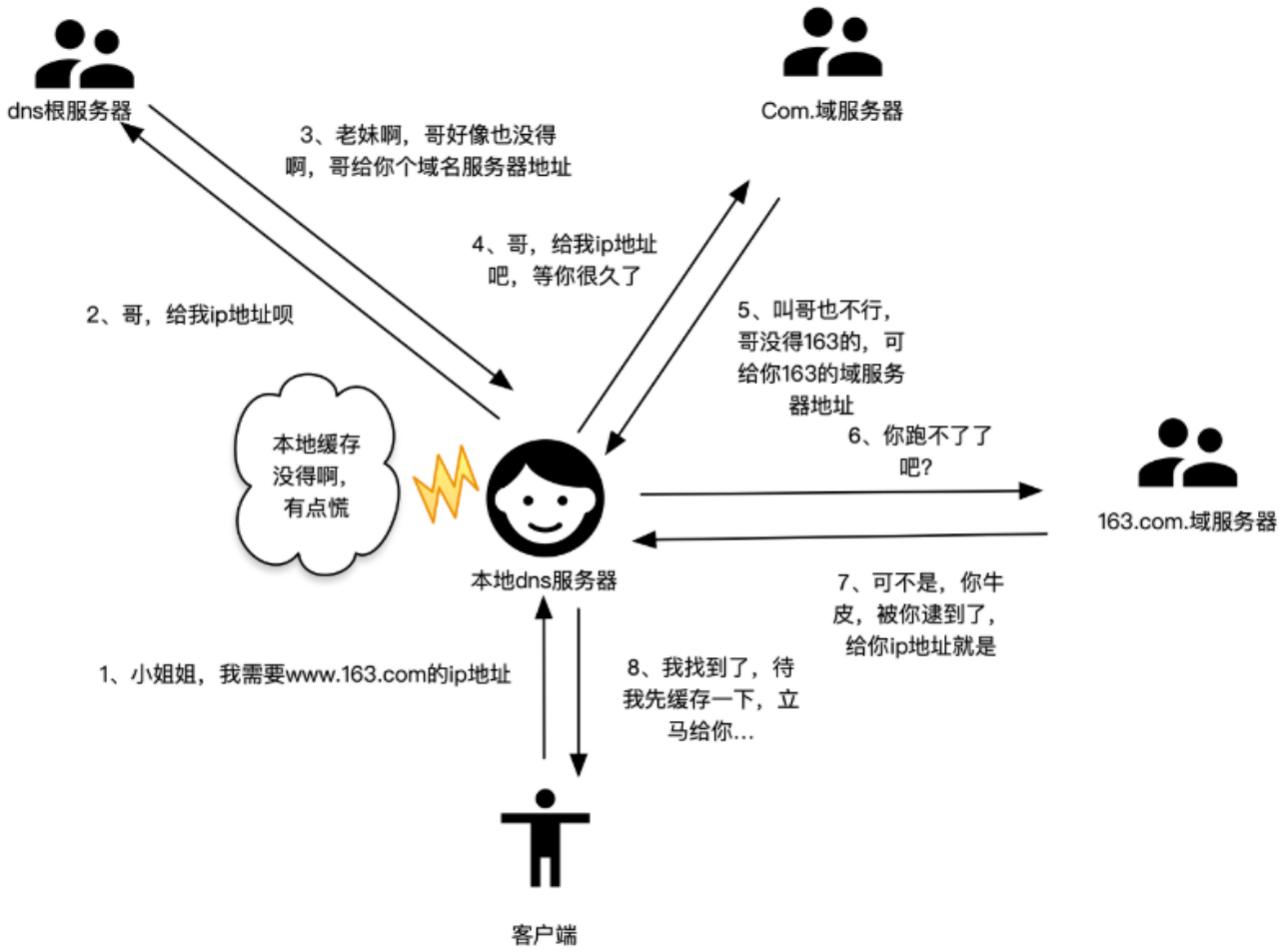
查询本地 DNS 服务器

如果 DNS 服务器和我们的主机在同一个子网内，系统会按照下面的 ARP 过程对 DNS 服务器进行 ARP查询

如果 DNS 服务器和我们的主机在不同的子网，系统会按照下面的 ARP 过程对默认网关进行查询

> 参考：https://www.zhihu.com/question/34873227/answer/518086565

17、谈谈DNS解析过程，具体一点



- 请求一旦发起，若是chrome浏览器，先在浏览器找之前**有没有缓存过的域名所对应的ip地址**，有的话，直接跳过dns解析了，若是没有，就会**找硬盘的hosts文件**，看看有没有，有的话，直接找到hosts文件里面的ip
- 如果本地的hosts文件没有能的到对应的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，中国移动等。
- 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。
- 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。

18、DNS负载均衡是什么策略?

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

19、HTTPS和HTTP的区别

1、HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全， HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

2、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。
3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

> 参考：https://www.cnblogs.com/wqhwe/p/5407468.html

20、什么是SSL/TLS？

SSL代表安全套接字层。它是一种用于加密和验证应用程序（如浏览器）和Web服务器之间发送的数据的协议。 身份验证， 加密Https的加密机制是一种共享密钥加密和公开密钥加密并用的混合加密机制。

SSL/TLS协议作用：认证用户和服务，加密数据，维护数据的完整性的应用层协议加密和解密需要两个不同的密钥，故被称为非对称加密；加密和解密都使用同一个密钥的

对称加密：优点在于加密、解密效率通常比较高 ，HTTPS 是基于非对称加密的， 公钥是公开的，

21、HTTPS是如何保证数据传输的安全，整体的流程是什么？（SSL是怎么工作保证安全的）

（1）客户端向服务器端发起SSL连接请求；
（2） 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
（3）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
（4）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，
（5）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。

因为数字签名、摘要是证书防伪非常关键的武器。“摘要”就是对传输的内容，通过hash算法计算出一段固定长度的串。然后，在通过CA的私钥对这段摘要进行加密，加密后得到的结果就是“数字签名”

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。

22、如何保证公钥不被篡改?

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。
（1） 客户端向服务器端索要并验证公钥。
（2） 双方协商生成"对话密钥"。
（3） 双方采用"对话密钥"进行加密通信。上面过程的前两步，又称为"握手阶段"（handshake）。

23、HTTP请求和响应报文有哪些主要字段?

请求报文

简单来说：

- 请求行：Request Line
- 请求头：Request Headers
- 请求体：Request Body

响应报文

简单来说：

- 状态行：Status Line
- 响应头：Response Headers
- 响应体：Response Body

24、Cookie是什么?

HTTP 协议是**无状态**的，主要是为了让 HTTP 协议尽可能简单，使得它能够处理大量事务，HTTP/1.1 引入 Cookie 来保存状态信息。

Cookie 是**服务器发送到用户浏览器并保存在本地的一小块数据**，它会在浏览器之后向同一服务器再次发起请求时被携带上，用于告知服务端两个请求是否来自同一浏览器。由于之后每次请求都会需要携带 Cookie 数据，因此会带来额外的性能开销（尤其是在移动环境下）。

Cookie 曾一度用于客户端数据的存储，因为当时并没有其它合适的存储办法而作为唯一的存储手段，但现在随着现代浏览器开始支持各种各样的存储方式，Cookie 渐渐被淘汰。

新的浏览器 API 已经允许开发者直接将数据存储到本地，如使用 Web storage API（本地存储和会话存储）或 IndexedDB。

cookie 的出现是因为 HTTP 是无状态的一种协议，换句话说，服务器记不住你，可能你每刷新一次网页，就要重新输入一次账号密码进行登录。这显然是让人无法接受的，cookie 的作用就好比服务器给你贴个标签，然后你每次向服务器再发请求时，服务器就能够 cookie 认出你。

抽象地概括一下：一个 cookie 可以认为是一个「变量」，形如 name=value，存储在浏览器；一个 session 可以理解为一种数据结构，多数情况是「映射」（键值对），存储在服务器上。

25、Cookie有什么用途?

- 会话状态管理（如用户登录状态、购物车、游戏分数或其它需要记录的信息）
- 个性化设置（如用户自定义设置、主题等）
- 浏览器行为跟踪（如跟踪分析用户行为等）

26、Session知识大总结

除了可以将用户信息通过 Cookie 存储在用户浏览器中，也可以利用 Session 存储在服务器端，存储在服务器端的信息更加安全。

Session 可以存储在服务器上的文件、数据库或者内存中。也可以将 Session 存储在 Redis 这种内存型数据库中，效率会更高。

使用 Session 维护用户登录状态的过程如下：

1. 用户进行登录时，用户提交包含用户名和密码的表单，放入 HTTP 请求报文中；
2. 服务器验证该用户名和密码，如果正确则把用户信息存储到 Redis 中，它在 Redis 中的 Key 称为 Session ID；
3. 服务器返回的响应报文的 Set-Cookie 首部字段包含了这个 Session ID，客户端收到响应报文之后将该 Cookie 值存入浏览器中；
4. 客户端之后对同一个服务器进行请求时会包含该 Cookie 值，服务器收到之后提取出 Session ID，从 Redis 中取出用户信息，继续之前的业务操作。

> 注意：Session ID 的安全性问题，不能让它被恶意攻击者轻易获取，那么就不能产生一个容易被猜到的 Session ID 值。此外，还需要经常重新生成 Session ID。在对安全性要求极高的场景下，例如转账等操作，除了使用 Session 管理用户状态之外，还需要对用户进行重新验证，比如重新输入密码，或者使用短信验证码等方式。

27、Session 的工作原理是什么?

session 的工作原理是客户端登录完成之后，服务器会创建对应的 session，session 创建完之后，会把 session 的 id 发送给客户端，客户端再存储到浏览器中。这样客户端每次访问服务器时，都会带着 sessionid，服务器拿到 sessionid 之后，在内存找到与之对应的 session 这样就可以正常工作了。

28、Cookie与Session的对比

HTTP作为无状态协议，必然需要在某种方式保持连接状态。这里简要介绍一下Cookie和Session。

- **Cookie**

  Cookie是客户端保持状态的方法。

  Cookie简单的理解就是存储由服务器发至客户端并由客户端保存的一段字符串。为了保持会话，服务器可以在响应客户端请求时将Cookie字符串放在Set-Cookie下，客户机收到Cookie之后保存这段字符串，之后再请求时候带上Cookie就可以被识别。

  除了上面提到的这些，Cookie在客户端的保存形式可以有两种，一种是会话Cookie一种是持久Cookie，会话Cookie就是将服务器返回的Cookie字符串保持在内存中，关闭浏览器之后自动销毁，持久Cookie则是存储在客户端磁盘上，其有效时间在服务器响应头中被指定，在有效期内，客户端再次请求服务器时都可以直接从本地取出。需要说明的是，存储在磁盘中的Cookie是可以被多个浏览器代理所共享的。

- **Session**

  Session是服务器保持状态的方法。

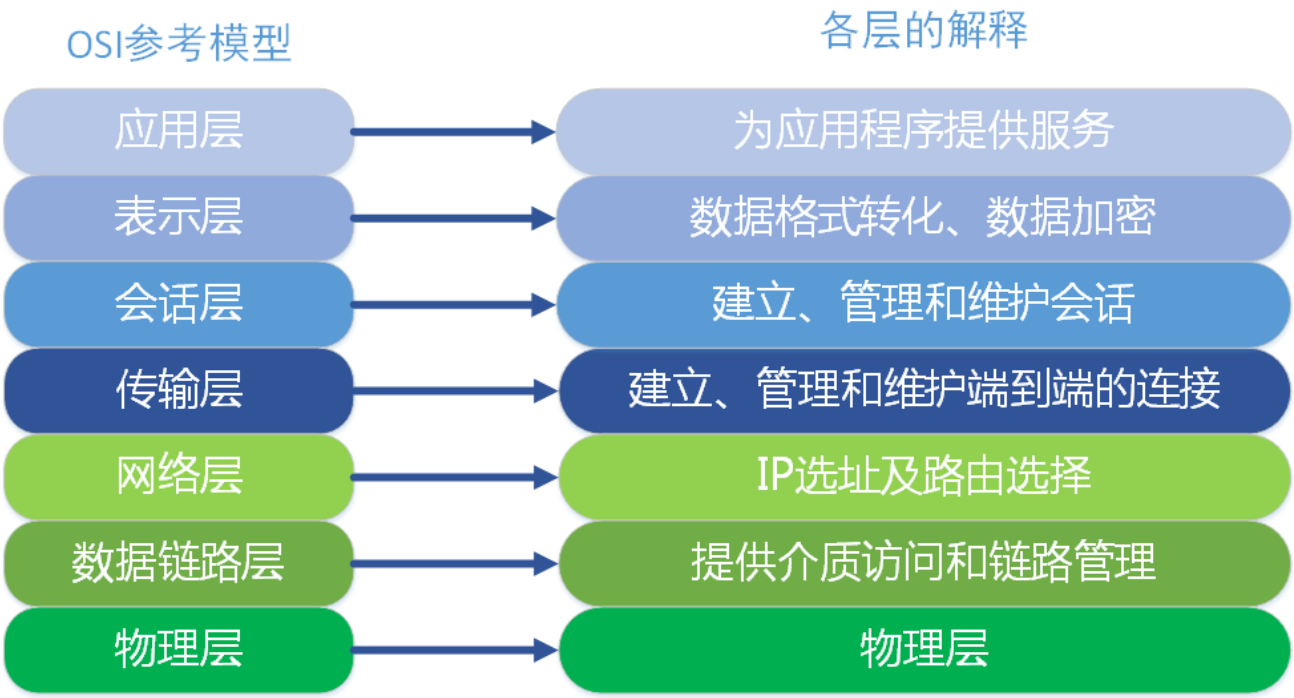
  首先需要明确的是，Session保存在服务器上，可以保存在数据库、文件或内存中，每个用户有独立的Session用户在客户端上记录用户的操作。我们可以理解为每个用户有一个独一无二的Session ID作为Session文件的Hash键，通过这个值可以锁定具体的Session结构的数据，这个Session结构中存储了用户操作行为。

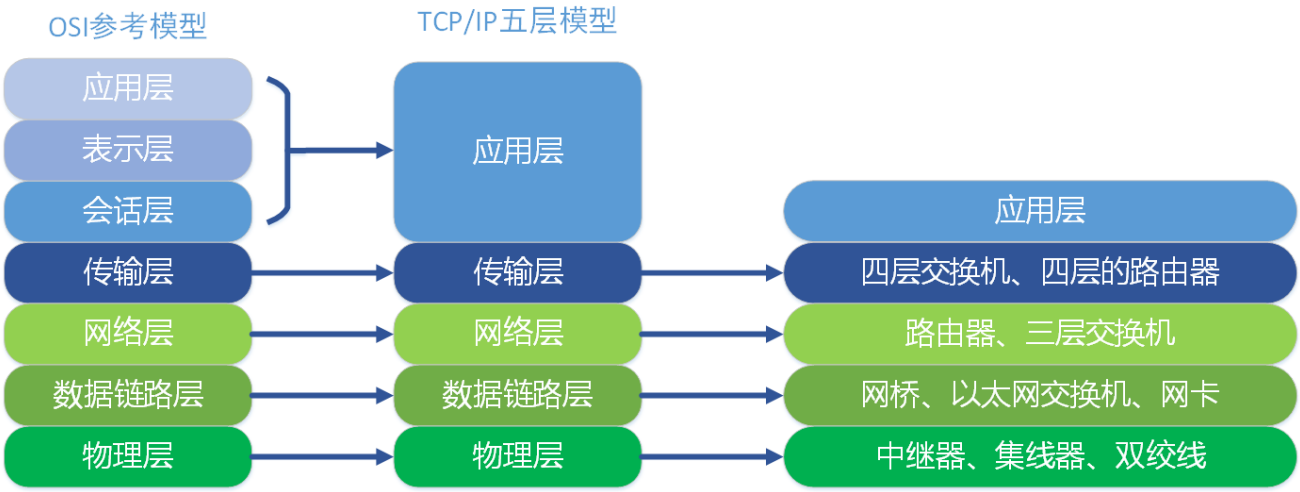
当服务器需要识别客户端时就需要结合Cookie了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用Cookie来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在Cookie里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。如果客户端的浏览器禁用了Cookie，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如sid=xxxxx这样的参数，服务端据此来识别用户，这样就可以帮用户完成诸如用户名等信息自动填入的操作了。

29、SQL注入攻击了解吗?

攻击者在HTTP请求中注入恶意的SQL代码，服务器使用参数构建数据库SQL命令时，恶意SQL被一起构造，并在数据库中执行。
用户登录，输入用户名 lianggzone，密码 ‘ or ‘1’=’1 ，如果此时使用参数构造的方式，就会出现
select * from user where name = ‘lianggzone’ and password = ‘’ or ‘1’=‘1’
不管用户名和密码是什么内容，使查询出来的用户列表不为空。如何防范SQL注入攻击使用预编译的PrepareStatement是必须的，但是一般我们会从两个方面同时入手。
Web端
1）有效性检验。
2）限制字符串输入的长度。
服务端
1）不用拼接SQL字符串。
2）使用预编译的PrepareStatement。
3）有效性检验。(为什么服务端还要做有效性检验？第一准则，外部都是不可信的，防止攻击者绕过Web端请求)
4）过滤SQL需要的参数中的特殊字符。比如单引号、双引号。

30、网络的七层模型与各自的功能（图片版）





31、什么是RARP？工作原理

概括： 反向地址转换协议，网络层协议，RARP与ARP工作方式相反。 RARP使只知道自己硬件地址的主机能够知道其IP地址。RARP发出要反向解释的物理地址并希望返回其IP地址，应答包括能够提供所需信息的RARP服务器发出的IP地址。
原理：
(1)网络上的每台设备都会有一个独一无二的硬件地址，通常是由设备厂商分配的MAC地址。主机从网卡上读取MAC地址，然后在网络上发送一个RARP请求的广播数据包，请求RARP服务器回复该主机的IP地址。

(2)RARP服务器收到了RARP请求数据包，为其分配IP地址，并将RARP回应发送给主机。

(3)PC1收到RARP回应后，就使用得到的IP地址进行通讯。

32、端口有效范围是多少到多少？

0-1023为知名端口号，比如其中HTTP是80，FTP是20（数据端口）、21（控制端口）

UDP和TCP报头使用两个字节存放端口号，所以端口号的有效范围是从0到65535。动态端口的范围是从1024到65535

33、为何需要把 TCP/IP 协议栈分成 5 层（或7层）？开放式回答。

答：ARPANET 的研制经验表明，对于复杂的计算机网络协议，其结构应该是层次式的。

分层的好处：

①隔层之间是独立的

②灵活性好

③结构上可以分隔开

④易于实现和维护

⑤能促进标准化工作。

34、DNS查询方式有哪些?

递归解析

当局部DNS服务器自己不能回答客户机的DNS查询时，它就需要向其他DNS服务器进行查询。此时有两种方式。局部DNS服务器自己负责向其他DNS服务器进行查询，一般是先向该域名的根域服务器查询，再由根域名服务器一级级向下查询。最后得到的查询结果返回给局部DNS服务器，再由局部DNS服务器返回给客户端。

迭代解析

当局部DNS服务器自己不能回答客户机的DNS查询时，也可以通过迭代查询的方式进行解析。局部DNS服务器不是自己向其他DNS服务器进行查询，而是把能解析该域名的其他DNS服务器的IP地址返回给客户端DNS程序，客户端DNS程序再继续向这些DNS服务器进行查询，直到得到查询结果为止。

也就是说，迭代解析只是帮你找到相关的服务器而已，而不会帮你去查。比如说：baidu.com的服务器ip地址在192.168.4.5这里，你自己去查吧，本人比较忙，只能帮你到这里了。

35、HTTP中缓存的私有和共有字段？知道吗?

private 指令规定了将资源作为私有缓存，只能被单独用户使用，一般存储在用户浏览器中。

```
Cache-Control: private
```

public 指令规定了将资源作为公共缓存，可以被多个用户使用，一般存储在代理服务器中。

```
Cache-Control: public
```

36、GET 方法参数写法是固定的吗?

在约定中，我们的参数是写在 ? 后面，用 & 分割。

我们知道，解析报文的过程是通过获取 TCP 数据，用正则等工具从数据中获取 Header 和 Body，从而提取参数。

比如header请求头中添加token，来验证用户是否登录等权限问题。

也就是说，我们可以自己约定参数的写法，只要服务端能够解释出来就行，万变不离其宗。

37、GET 方法的长度限制是怎么回事?

网络上都会提到浏览器地址栏输入的参数是有限的。

首先说明一点，HTTP 协议没有 Body 和 URL 的长度限制，对 URL 限制的大多是浏览器和服务器的原因。

浏览器原因就不说了，服务器是因为处理长 URL 要消耗比较多的资源，为了性能和安全（防止恶意构造长 URL 来攻击）考虑，会给 URL 长度加限制。

38、POST 方法比 GET 方法安全?

有人说POST 比 GET 安全，因为数据在地址栏上不可见。

然而，从传输的角度来说，他们都是不安全的，因为 HTTP 在网络上是明文传输的，只要在网络节点上捉包，就能完整地获取数据报文。

要想安全传输，就只有加密，也就是 HTTPS。

39、POST 方法会产生两个 TCP 数据包？你了解吗?

有些文章中提到，POST 会将 header 和 body 分开发送，先发送 header，服务端返回 100 状态码再发送 body。

HTTP 协议中没有明确说明 POST 会产生两个 TCP 数据包，而且实际测试(Chrome)发现，header 和 body 不会分开发送。

所以，header 和 body 分开发送是部分浏览器或框架的请求方法，不属于 post 必然行为。

40、Session是什么?

除了可以将用户信息通过 Cookie 存储在用户浏览器中，也可以利用 Session 存储在服务器端，存储在服务器端的信息更加安全。

Session 可以存储在服务器上的文件、数据库或者内存中。也可以将 Session 存储在 Redis 这种内存型数据库中，效率会更高。

41、使用 Session 的过程是怎样的?

过程如下：

- 用户进行登录时，用户提交包含用户名和密码的表单，放入 HTTP 请求报文中；
- 服务器验证该用户名和密码，如果正确则把用户信息存储到 Redis 中，它在 Redis 中的 Key 称为 Session ID；
- 服务器返回的响应报文的 Set-Cookie 首部字段包含了这个 Session ID，客户端收到响应报文之后将该 Cookie 值存入浏览器中；
- 客户端之后对同一个服务器进行请求时会包含该 Cookie 值，服务器收到之后提取出 Session ID，从 Redis 中取出用户信息，继续之前的业务操作。

注意：Session ID 的安全性问题，不能让它被恶意攻击者轻易获取，那么就不能产生一个容易被猜到的 Session ID 值。此外，还需要经常重新生成 Session ID。在对安全性要求极高的场景下，例如转账等操作，除了使用 Session 管理用户状态之外，还需要对用户进行重新验证，比如重新输入密码，或者使用短信验证码等方式。

42、Session和cookie应该如何去选择（适用场景）？

- Cookie 只能存储 ASCII 码字符串，而 Session 则可以存储任何类型的数据，因此在考虑数据复杂性时首选 Session；
- Cookie 存储在浏览器中，容易被恶意查看。如果非要将一些隐私数据存在 Cookie 中，可以将 Cookie 值进行加密，然后在服务器进行解密；
- 对于大型网站，如果用户所有的信息都存储在 Session 中，那么开销是非常大的，因此不建议将所有的用户信息都存储到 Session 中。

43、Cookies和Session区别是什么?

Cookie和Session都是客户端与服务器之间保持状态的解决方案
1，存储的位置不同，cookie：存放在客户端，session：存放在服务端。Session存储的数据比较安全
2，存储的数据类型不同
两者都是key-value的结构，但针对value的类型是有差异的
cookie：value只能是字符串类型，session：value是Object类型
3，存储的数据大小限制不同
cookie：大小受浏览器的限制，很多是是4K的大小， session：理论上受当前内存的限制，
4，生命周期的控制
cookie的生命周期当浏览器关闭的时候，就消亡了
(1)cookie的生命周期是累计的，从创建时，就开始计时，20分钟后，cookie生命周期结束，
(2)session的生命周期是间隔的，从创建时，开始计时如在20分钟，没有访问session，那么session生命周期被销毁

44、DDos 攻击了解吗?

客户端向服务端发送请求链接数据包，服务端向客户端发送确认数据包，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
没有彻底根治的办法，除非不使用TCP
DDos 预防：
1）限制同时打开SYN半链接的数目
2）缩短SYN半链接的Time out 时间
3）关闭不必要的服务

45、MTU和MSS分别是什么?

MTU：maximum transmission unit，最大传输单元，由硬件规定，如以太网的MTU为1500字节。

MSS：maximum segment size，最大分节大小，为TCP数据包每次传输的最大数据分段大小，一般由发送端向对端TCP通知对端在每个分节中能发送的最大TCP数据。MSS值为MTU值减去IPv4 Header（20 Byte）和TCP header（20 Byte）得到。

46、HTTP中有个缓存机制，但如何保证缓存是最新的呢？（缓存过期机制）

max-age 指令出现在请求报文，并且缓存资源的缓存时间小于该指令指定的时间，那么就能接受该缓存。

max-age 指令出现在响应报文，表示缓存资源在缓存服务器中保存的时间。

```
Cache-Control: max-age=31536000
```

Expires 首部字段也可以用于告知缓存服务器该资源什么时候会过期。

```
Expires: Wed, 04 Jul 2012 08:26:05 GMT
```

- 在 HTTP/1.1 中，会优先处理 max-age 指令；
- 在 HTTP/1.0 中，max-age 指令会被忽略掉。

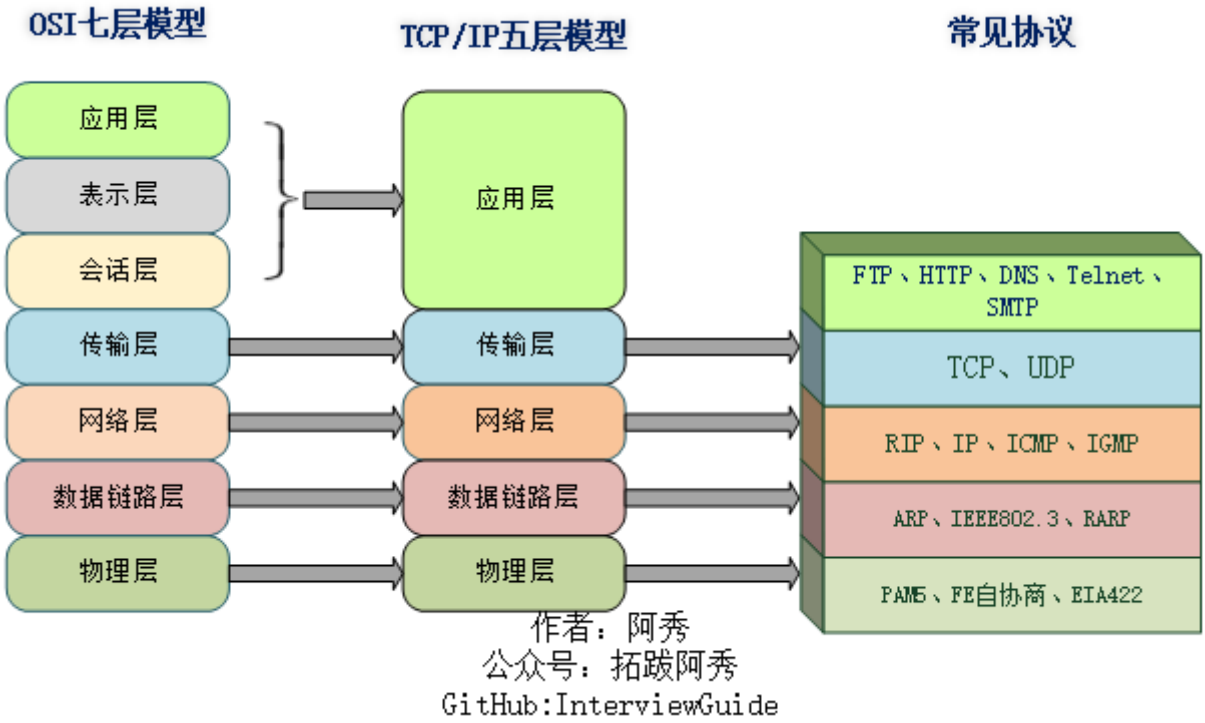
47、TCP头部中有哪些信息?

- 序号（32bit）：传输方向上字节流的字节编号。初始时序号会被设置一个随机的初始值（ISN），之后每次发送数据时，序号值 = ISN + 数据在整个字节流中的偏移。假设A -> B且ISN = 1024，第一段数据512字节已经到B，则第二段数据发送时序号为1024 + 512。用于解决网络包乱序问题。
- 确认号（32bit）：接收方对发送方TCP报文段的响应，其值是收到的序号值 + 1。
- 首部长（4bit）：标识首部有多少个4字节 * 首部长，最大为15，即60字节。
- 标志位（6bit）：
  - URG：标志紧急指针是否有效。
  - ACK：标志确认号是否有效（确认报文段）。用于解决丢包问题。
  - PSH：提示接收端立即从缓冲读走数据。
  - RST：表示要求对方重新建立连接（复位报文段）。
  - SYN：表示请求建立一个连接（连接报文段）。
  - FIN：表示关闭连接（断开报文段）。
- 窗口（16bit）：接收窗口。用于告知对方（发送方）本方的缓冲还能接收多少字节数据。用于解决流控。
- 校验和（16bit）：接收端用CRC检验整个报文段有无损坏。

48、常见TCP的连接状态有哪些?

- CLOSED：初始状态。
- LISTEN：服务器处于监听状态。
- SYN_SEND：客户端socket执行CONNECT连接，发送SYN包，进入此状态。
- SYN_RECV：服务端收到SYN包并发送服务端SYN包，进入此状态。
- ESTABLISH：表示连接建立。客户端发送了最后一个ACK包后进入此状态，服务端接收到ACK包后进入此状态。
- FIN_WAIT_1：终止连接的一方（通常是客户机）发送了FIN报文后进入。等待对方FIN。
- CLOSE_WAIT：（假设服务器）接收到客户机FIN包之后等待关闭的阶段。在接收到对方的FIN包之后，自然是需要立即回复ACK包的，表示已经知道断开请求。但是本方是否立即断开连接（发送FIN包）取决于是否还有数据需要发送给客户端，若有，则在发送FIN包之前均为此状态。
- FIN_WAIT_2：此时是半连接状态，即有一方要求关闭连接，等待另一方关闭。客户端接收到服务器的ACK包，但并没有立即接收到服务端的FIN包，进入FIN_WAIT_2状态。
- LAST_ACK：服务端发动最后的FIN包，等待最后的客户端ACK响应，进入此状态。
- TIME_WAIT：客户端收到服务端的FIN包，并立即发出ACK包做最后的确认，在此之后的2MSL时间称为TIME_WAIT状态。

49、网络的七层/五层模型主要的协议有哪些?



50、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。

51、TCP头部报文字段介绍几个？各自的功能?

source port 和 destination port

> 两者分别为「源端口号」和「目的端口号」。源端口号就是指本地端口，目的端口就是远程端口。

可以这么理解，我们有很多软件，每个软件都对应一个端口，假如，你想和我数据交互，咱们得互相知道你我的端口号。

再来一个很官方的：

> 扩展：应用程序的端口号和应用程序所在主机的 IP 地址统称为 socket（套接字），IP:端口号, 在互联网上 socket 唯一标识每一个应用程序，源端口+源IP+目的端口+目的IP称为”套接字对“，一对套接字就是一个连接，一个客户端与服务器之间的连接。

Sequence Number

> 称为「序列号」。用于 TCP 通信过程中某一传输方向上字节流的每个字节的编号，为了确保数据通信的有序性，避免网络中乱序的问题。接收端根据这个编号进行确认，保证分割的数据段在原始数据包的位置。初始序列号由自己定，而后续的序列号由对端的 ACK 决定：SN_x = ACK_y (x 的序列号 = y 发给 x 的 ACK)。

说白了，类似于身份证一样，而且还得发送此时此刻的所在的位置，就相当于身份证上的地址一样。

Acknowledge Number

> 称为「确认序列号」。确认序列号是接收确认端所期望收到的下一序列号。确认序号应当是上次已成功收到数据字节序号加1，只有当标志位中的 ACK 标志为 1 时该确认序列号的字段才有效。主要用来解决不丢包的问题。

TCP Flag

TCP 首部中有 6 个标志比特，它们中的多个可同时被设置为 1，主要是用于操控 TCP 的状态机的，依次为URG，ACK，PSH，RST，SYN，FIN。

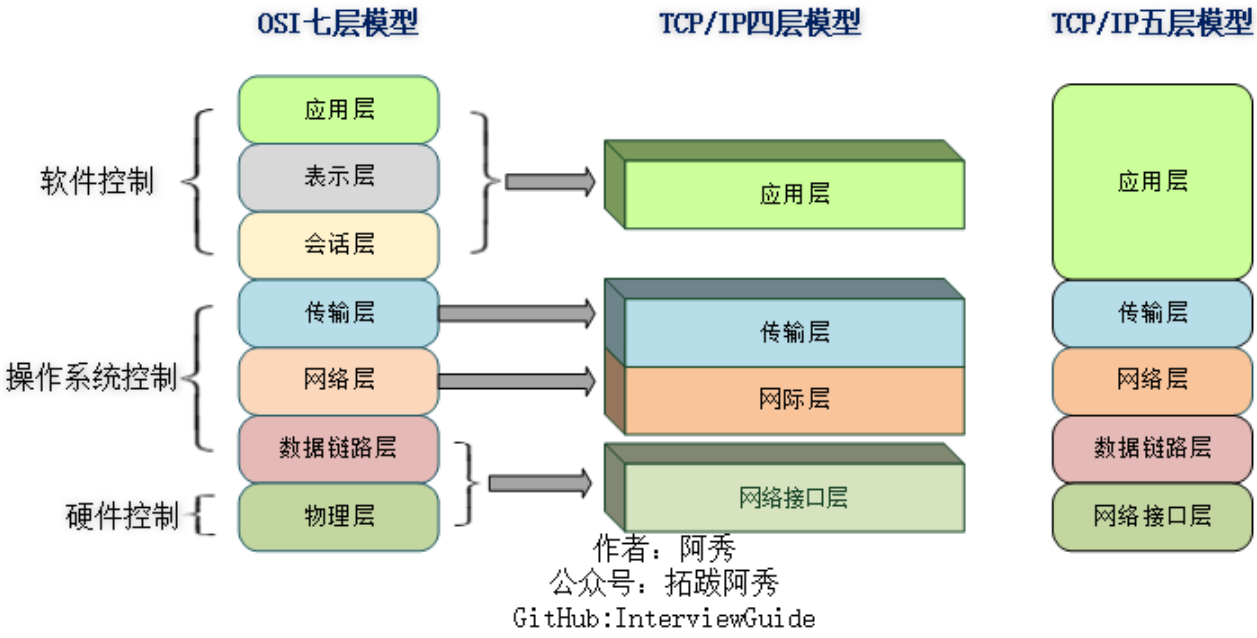
当然只介绍三个：

1. **ACK**：这个标识可以理解为发送端发送数据到接收端，发送的时候 ACK 为 0，标识接收端还未应答，一旦接收端接收数据之后，就将 ACK 置为 1，发送端接收到之后，就知道了接收端已经接收了数据。
2. **SYN**：表示「同步序列号」，是 TCP 握手的发送的第一个数据包。用来建立 TCP 的连接。SYN 标志位和 ACK 标志位搭配使用，当连接请求的时候，SYN=1，ACK=0连接被响应的时候，SYN=1，ACK=1；这个标志的数据包经常被用来进行端口扫描。扫描者发送一个只有 SYN 的数据包，如果对方主机响应了一个数据包回来 ，就表明这台主机存在这个端口。
3. **FIN**：表示发送端已经达到数据末尾，也就是说双方的数据传送完成，没有数据可以传送了，发送FIN标志位的 TCP 数据包后，连接将被断开。这个标志的数据包也经常被用于进行端口扫描。发送端只剩最后的一段数据了，同时要告诉接收端后边没有数据可以接受了，所以用FIN标识一下，接收端看到这个FIN之后，哦！这是接受的最后的数据，接受完就关闭了；**TCP四次分手必然问。**

Window size

> 称为滑动窗口大小。所说的滑动窗口，用来进行流量控制。

52、OSI 的七层模型的主要功能?



物理层：利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。
数据链路层：接收来自物理层的位流形式的数据，并封装成帧，传送到上一层
网络层：将网络地址翻译成对应的物理地址，并通过路由选择算法为分组通过通信子网选择最适当的路径。
传输层：在源端与目的端之间提供可靠的透明数据传输
会话层：负责在网络中的两节点之间建立、维持和终止通信
表示层：处理用户信息的表示问题，数据的编码，压缩和解压缩，数据的加密和解密
应用层：为用户的应用进程提供网络通信服务

53、应用层常见协议知道多少？了解几个？

协议	名称	默认端口	底层协议
HTTP	超文本传输协议	80	TCP
HTTPS	超文本传输安全协议	443	TCP
Telnet	远程登录服务的标准协议	23	TCP
FTP	文件传输协议	20传输和21连接	TCP
TFTP	简单文件传输协议	21	UDP
SMTP	简单邮件传输协议（发送用）	25	TCP
POP	邮局协议（接收用）	110	TCP
DNS	域名解析服务	53	服务器间进行域传输的时候用TCP 客户端查询DNS服务器时用 UDP

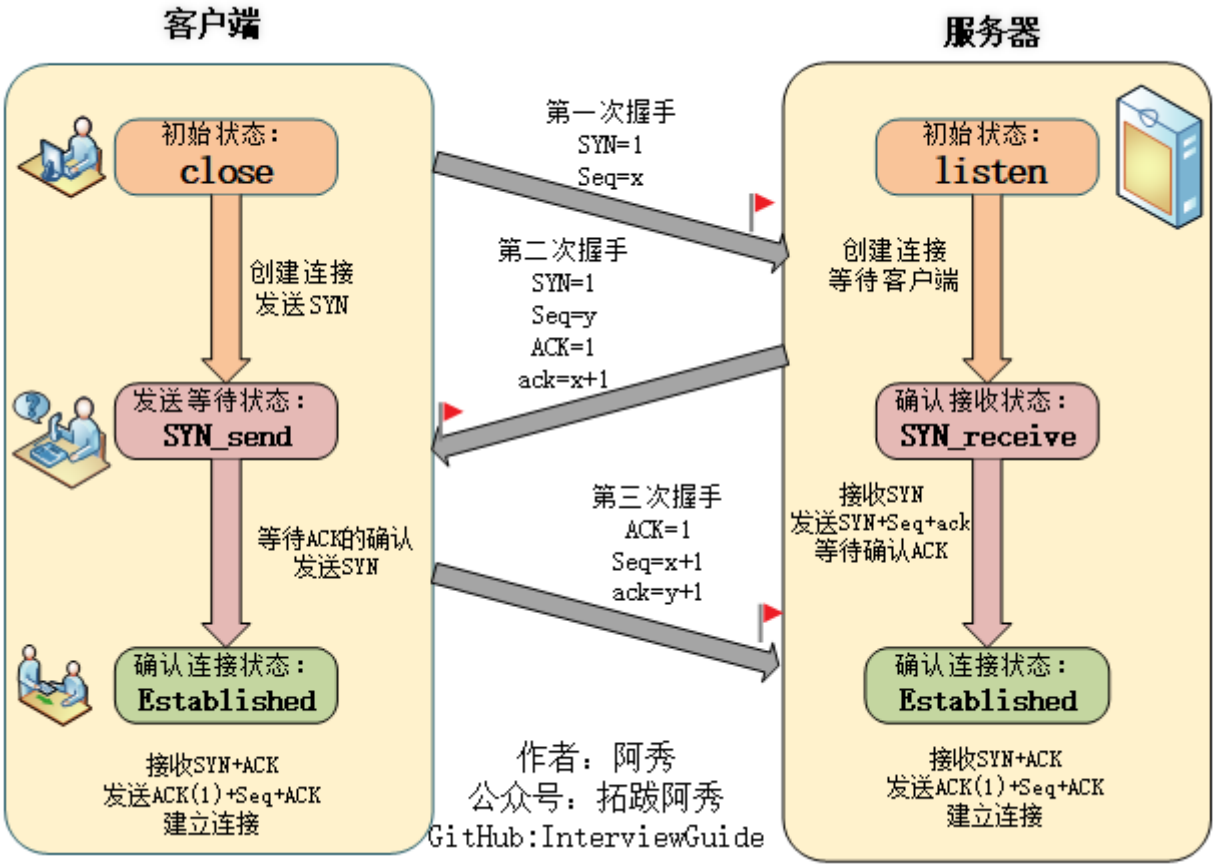
54、浏览器在与服务器建立了一个 TCP 连接后是否会在一个 HTTP 请求完成后断开？什么情况下会断开？

在 HTTP/1.0 中，一个服务器在发送完一个 HTTP 响应后，会断开 TCP 链接。但是这样每次请求都会重新建立和断开 TCP 连接，代价过大。所以虽然标准中没有设定，**某些服务器对 Connection: keep-alive 的 Header 进行了支持**。意思是说，完成这个 HTTP 请求之后，不要断开 HTTP 请求使用的 TCP 连接。这样的好处是连接可以被重新使用，之后发送 HTTP 请求的时候不需要重新建立 TCP 连接，以及如果维持连接，那么 SSL 的开销也可以避免。

持久连接：既然维持 TCP 连接好处这么多，HTTP/1.1 就把 Connection 头写进标准，并且默认开启持久连接，除非请求中写明 Connection: close，那么浏览器和服务器之间是会维持一段时间的 TCP 连接，不会一个请求结束就断掉。

默认情况下建立 TCP 连接不会断开，只有在请求报头中声明 Connection: close 才会在请求完成后关闭连接。

55、三次握手相关内容



三次握手（Three-way Handshake）其实就是指建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用就是为了确认双方的接收能力和发送能力是否正常、指定自己的初始化序列号为后面的可靠性传送做准备。实质上其实就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。

第一种回答

刚开始客户端处于 Closed 的状态，服务端处于 Listen 状态，进行三次握手：

- 第一次握手：客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN(c)。此时客户端处于 SYN_SEND 状态。

  首部的同步位SYN=1，初始序号seq=x，SYN=1的报文段不能携带数据，但要消耗掉一个序号。
- 第二次握手：服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN(s)。同时会把客户端的 ISN + 1 作为ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 SYN_RCVD 的状态。

  在确认报文段中SYN=1，ACK=1，确认号ack=x+1，初始序号seq=y。
- 第三次握手：客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 ESTABLISHED 状态。服务器收到 ACK 报文之后，也处于 ESTABLISHED 状态，此时，双方已建立起了连接。

  确认报文段ACK=1，确认号ack=y+1，序号seq=x+1（初始为seq=x，第二个报文段所以要+1），ACK报文段可以携带数据，不携带数据则不消耗序号。

发送第一个SYN的一端将执行主动打开（active open），接收这个SYN并发回下一个SYN的另一端执行被动打开（passive open）。

在socket编程中，客户端执行connect()时，将触发三次握手。

第二种回答

- 初始状态：客户端处于 closed(关闭)状态，服务器处于 listen(监听) 状态。
- 第一次握手：客户端发送请求报文将 SYN = 1同步序列号和初始化序列号seq = x发送给服务端，发送完之后客户端处于SYN_Send状态。（验证了客户端的发送能力和服务端的接收能力）
- 第二次握手：服务端受到 SYN 请求报文之后，如果同意连接，会以自己的同步序列号SYN(服务端) = 1、初始化序列号 seq = y和确认序列号（期望下次收到的数据包）ack = x+ 1 以及确认号ACK = 1报文作为应答，服务器为SYN_Receive状态。（问题来了，两次握手之后，站在客户端角度上思考：我发送和接收都ok，服务端的发送和接收也都ok。但是站在服务端的角度思考：哎呀，我服务端接收ok，但是我不清楚我的发送ok不ok呀，而且我还不知道你接受能力如何呢？所以老哥，你需要给我三次握手来传个话告诉我一声。你要是不告诉我，万一我认为你跑了，然后我可能出于安全性的考虑继续给你发一次，看看你回不回我。）
- 第三次握手： 客户端接收到服务端的 SYN + ACK之后，知道可以下次可以发送了下一序列的数据包了，然后发送同步序列号 ack = y + 1和数据包的序列号 seq = x + 1以及确认号ACK = 1确认包作为应答，客户端转为established状态。（分别站在双方的角度上思考，各自ok）

56、为什么需要三次握手，两次不行吗?

弄清这个问题，我们需要先弄明白三次握手的目的是什么，能不能只用两次握手来达到同样的目的。

- 第一次握手：客户端发送网络包，服务端收到了。 这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的。
- 第二次握手：服务端发包，客户端收到了。 这样客户端就能得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。不过此时服务器并不能确认客户端的接收能力是否正常。
- 第三次握手：客户端发包，服务端收到了。 这样服务端就能得出结论：客户端的接收、发送能力正常，服务器自己的发送、接收能力也正常。

因此，需要三次握手才能确认双方的接收与发送能力是否正常。

试想如果是用两次握手，则会出现下面这种情况：

> 如客户端发出连接请求，但因连接请求报文丢失而未收到确认，于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接，客户端共发出了两个连接请求报文段，其中第一个丢失，第二个到达了服务端，但是第一个丢失的报文段只是在**某些网络结点长时间滞留了，延误到连接释放以后的某个时间才到达服务端**，此时服务端误认为客户端又发出一次新的连接请求，于是就向客户端发出确认报文段，同意建立连接，不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端忽略服务端发来的确认，也不发送数据，则服务端一致等待客户端发送数据，浪费资源。

57、什么是半连接队列?

服务器第一次收到客户端的 SYN 之后，就会处于 SYN_RCVD 状态，此时双方还没有完全建立其连接，服务器会把此种状态下请求连接放在一个**队列**里，我们把这种队列称之为**半连接队列**。

当然还有一个**全连接队列**，就是已经完成三次握手，建立起连接的就会放在全连接队列中。如果队列满了就有可能会出现丢包现象。

这里在补充一点关于**SYN-ACK 重传次数**的问题： 服务器发送完SYN-ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传。如果重传次数超过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。 注意，每次重传等待的时间不一定相同，一般会是指数增长，例如间隔时间为 1s，2s，4s，8s.....

58、 ISN(Initial Sequence Number)是固定的吗?

当一端为建立连接而发送它的SYN时，它为连接选择一个初始序号。ISN随时间而变化，因此每个连接都将具有不同的ISN。ISN可以看作是一个32比特的计数器，每4ms加1 。这样选择序号的目的在于防止在网络中被延迟的分组在以后又被传送，而导致某个连接的一方对它做错误的解释。

三次握手的其中一个重要功能是客户端和服务端交换 ISN(Initial Sequence Number)，以便让对方知道接下来接收数据的时候如何按序列号组装数据。如果 ISN 是固定的，攻击者很容易猜出后续的确认号，因此 ISN 是动态生成的。

59、 三次握手过程中可以携带数据吗?

其实第三次握手的时候，是可以携带数据的。但是，**第一次、第二次握手不可以携带数据**

为什么这样呢?大家可以想一个问题，假如第一次握手可以携带数据的话，如果有人要恶意攻击服务器，那他每次都在第一次握手中的 SYN 报文中放入大量的数据。因为攻击者根本就不理服务器的接收、发送能力是否正常，然后疯狂着重复发 SYN 报文的话，这会让服务器花费很多时间、内存空间来接收这些报文。

也就是说，**第一次握手不可以放数据，其中一个简单的原因就是会让服务器更加容易受到攻击了。而对于第三次的话，此时客户端已经处于 ESTABLISHED 状态。对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据也没啥毛病。**

60、SYN攻击是什么?

服务器端的资源分配是在二次握手时分配的，而客户端的资源是在完成三次握手时分配的，所以服务器容易受到SYN洪泛攻击。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server则回复确认包，并等待Client确认，由于源地址不存在，因此Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络拥塞甚至系统瘫痪。SYN 攻击是一种典型的 DoS/DDoS 攻击。

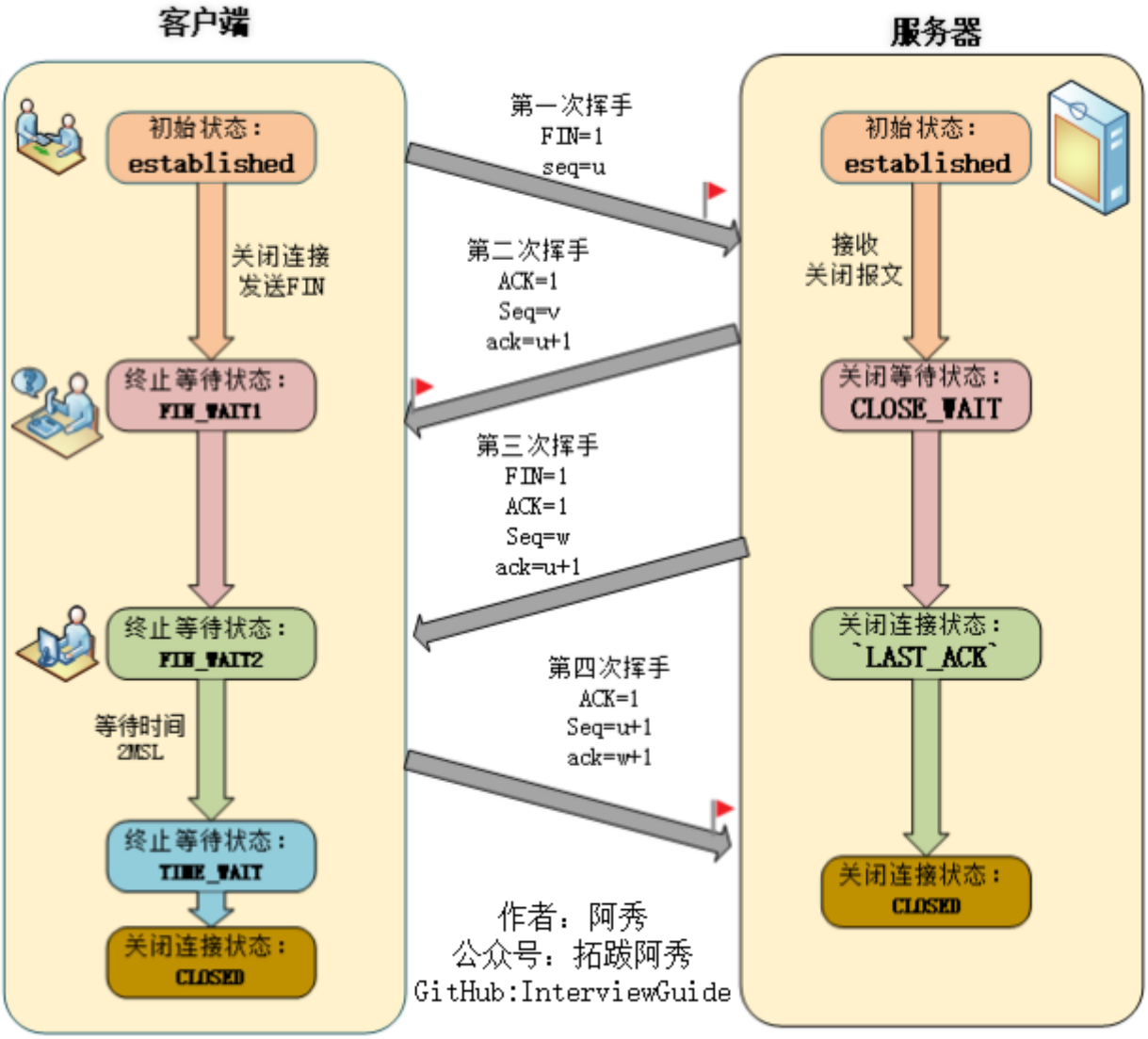
检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源IP地址是随机的，基本上可以断定这是一次SYN攻击。在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。

```
netstat -n -p TCP | grep SYN_RECV
复制代码
```

常见的防御 SYN 攻击的方法有如下几种：

- 缩短超时（SYN Timeout）时间
- 增加最大半连接数
- 过滤网关防护
- SYN cookies技术

61、 四次挥手相关内容



建立一个连接需要三次握手，而终止一个连接要经过四次挥手（也有将四次挥手叫做四次握手的）。这由TCP的**半关闭**（half-close）造成的。所谓的半关闭，其实就是TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力。

TCP 的连接的拆除需要发送四个包，因此称为四次挥手(Four-way handshake)，客户端或服务器均可主动发起挥手动作。

第一种回答

刚开始双方都处于 ESTABLISHED 状态，假如是客户端先发起关闭请求。四次挥手的过程如下：

- 第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于 `FIN_WAIT1` 状态。 即发出**连接释放报文段**（FIN=1，序号seq=u），并停止再发送数据，主动关闭TCP连接，进入FIN_WAIT1（终止等待1）状态，等待服务端的确认。
- 第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 +1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 `CLOSE_WAIT` 状态。 即服务端收到连接释放报文段后即发出**确认报文段**（ACK=1，确认号ack=u+1，序号seq=v），服务端进入CLOSE_WAIT（关闭等待）状态，此时的TCP处于半关闭状态，客户端到服务端的连接释放。客户端收到服务端的确认后，进入FIN_WAIT2（终止等待2）状态，等待服务端发出的连接释放报文段。
- 第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 `LAST_ACK` 的状态。 即服务端没有要向客户端发出的数据，服务端发出**连接释放报文段**（FIN=1，ACK=1，序号seq=w，确认号ack=u+1），服务端进入LAST_ACK（最后确认）状态，等待客户端的确认。
- 第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 +1 作为自己 ACK 报文的序列号值，此时客户端处于 `TIME_WAIT` 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态，服务端收到 ACK 报文之后，就处于关闭连接了，处于 `CLOSED` 状态。 即客户端收到服

务端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），客户端进入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后，客户端才进入CLOSED状态。

收到一个FIN只意味着在这一方向上没有数据流动。客户端执行主动关闭并进入TIME_WAIT是正常的，服务端通常执行被动关闭，不会进入TIME_WAIT状态。

在socket编程中，任何一方执行close()操作即可产生挥手操作。

第二种回答

- 初始化状态：客户端和服务端都在连接状态，接下来开始进行四次分手断开连接操作。
- 第一次分手：第一次分手无论是客户端还是服务端都可以发起，因为 TCP 是全双工的。

> 假如客户端发送的数据已经发送完毕，发送FIN = 1 告诉服务端，客户端所有数据已经全发完了，服务端你可以关闭接收了，但是如果你们服务端有数据要发给客户端，客户端照样可以接收的。此时客户端处于FIN = 1 等待服务端确认释放连接状态。

- 第二次分手：服务端接收到客户端的释放请求连接之后，知道客户端没有数据要发给自己了，然后服务端发送ACK = 1告诉客户端收到你发给我的信息，此时服务端处于 CLOSE_WAIT 等待关闭状态。（服务端先回应给客户端一声，我知道了，但服务端的发送数据能力即将等待关闭，于是接下来第三次就来了。）
- 第三次分手：此时服务端向客户端把所有的数据发送完了，然后发送一个FIN = 1，用于告诉客户端，服务端的所有数据发送完毕，客户端你也可以关闭接收数据连接了。此时服务端状态处于LAST_ACK状态，来等待确认客户端是否收到了自己的请求。（服务端等客户端回复是否收到呢，不收到的话，服务端不知道客户端是不是挂掉了还是咋回事呢，所以服务端不敢关闭自己的接收能力，于是第四次就来了。）
- 第四次分手：此时如果客户端收到了服务端发送完的信息之后，就发送ACK = 1，告诉服务端，客户端已经收到了你的信息。有一个 2 MSL 的延迟等待。

62、挥手为什么需要四次?

第一种回答

因为当服务端收到客户端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中**ACK报文是用来应答的，SYN报文是用来同步的**。但是关闭连接时，当服务端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉客户端，"你发的FIN报文我收到了"。只有等到我服务端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四次挥手。

第二种回答

任何一方都可以在数据传送结束后发出连接释放的通知，待对方确认后进入半关闭状态。当另一方也没有数据再发送的时候，则发出连接释放通知，对方确认后就完全关闭了TCP连接。举个例子：A 和 B 打电话，通话即将结束后，A 说“我没啥要说的了”，B回答“我知道了”，但是 B 可能还会有要说的话，A 不能要求 B 跟着自己的节奏结束通话，于是 B 可能又巴拉巴拉说了一通，最后 B 说“我说完了”，A 回答“知道了”，这样通话才算结束。

63、2MSL等待状态?

TIME_WAIT状态也成为2MSL等待状态。每个具体TCP实现必须选择一个报文段最大生存时间MSL（Maximum Segment Lifetime），它是任何报文段被丢弃前在网络内的最长时间。这个时间是有限的，因为TCP报文段以IP数据报在网络内传输，而IP数据报则有限制其生存时间的TTL字段。

对一个具体实现所给定的MSL值，处理的原则是：当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIME_WAIT状态停留的时间为2倍的MSL。这样可让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并重发最后的FIN）。

这种2MSL等待的另一个结果是这个TCP连接在2MSL等待期间，定义这个连接的插口（客户的IP地址和端口号，服务器的IP地址和端口号）不能再被使用。这个连接只能在2MSL结束后才能再被使用。

64、四次挥手释放连接时，等待2MSL的意义?

> **MSL**是Maximum Segment Lifetime的英文缩写，可译为“最长报文段寿命”，它是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。

为了保证客户端发送的最后一个ACK报文段能够到达服务器。因为这个ACK有可能丢失，从而导致处在LAST-ACK状态的服务器收不到对FIN-ACK的确认报文。服务器会超时重传这个FIN-ACK，接着客户端再重传一次确认，重新启动时间等待计时器。最后客户端和服务器都能正常的关闭。假设客户端不等待2MSL，而是在发送完ACK之后直接释放关闭，一但这个ACK丢失的话，服务器就无法正常的进入关闭连接状态。

两个理由

1. 保证客户端发送的最后一个ACK报文段能够到达服务端。 这个ACK报文段有可能丢失，使得处于LAST-ACK状态的B收不到对已发送的FIN+ACK报文段的确认，服务端超时重传FIN+ACK报文段，而客户端能在2MSL时间内收到这个重传的FIN+ACK报文段，接着客户端重传一次确认，重新启动2MSL计时器，最后客户端和服务端都进入到CLOSED状态，若客户端在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，则无法收到服务端重传的FIN+ACK报文段，所以不会再发送一次确认报文段，则服务端无法正常进入到CLOSED状态。
2. 防止“已失效的连接请求报文段”出现在本连接中。 客户端在发送完最后一个ACK报文段后，再经过2MSL，就可以使本连接持续的时间内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这种旧的连接请求报文段。

65、为什么TIME_WAIT状态需要经过2MSL才能返回到CLOSE状态?

第一种回答

理论上，四个报文都发送完毕，就可以直接进入CLOSE状态了，但是可能网络是不可靠的，有可能最后一个ACK丢失。所以TIME_WAIT状态就是用来重发可能丢失的ACK报文。

第二种回答

对应这样一种情况，最后客户端发送的ACK = 1给服务端的过程中丢失了，服务端没收到，服务端怎么认为的？我已经发送完数据了，怎么客户端没回应我？是不是中途丢失了？然后服务端再次发起断开连接的请求，一个来回就是2MSL。

客户端给服务端发送的ACK = 1丢失，服务端等待 1MSL没收到，然后重新发送消息需要1MSL。如果再次接收到服务端的消息，则重启2MSL计时器，发送确认请求。客户端只需等待2MSL，如果没有再次收到服务端的消息，就说明服务端已经接收到自己确认消息；此时双方都关闭的连接，TCP 四次分手完毕

66、TCP粘包问题是什么？你会如何去解决它?

TCP粘包是指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。

- 由TCP**连接复用**造成的粘包问题。
- 因为TCP默认会使用**Nagle算法**，此算法会导致粘包问题。
  - 只有上一个分组得到确认，才会发送下一个分组；
  - 收集多个小分组，在一个确认到来时一起发送。

- 数据包过大造成的粘包问题。
- 流量控制，拥塞控制也可能导致粘包。
- 接收方不及时接收缓冲区的包，造成多个包接收

解决：

1. **Nagle算法**问题导致的，需要结合应用场景适当关闭该算法
2. 尾部标记序列。通过特殊标识符表示数据包的边界，例如\n\r，\t，或者一些隐藏字符。
3. 头部标记分步接收。在TCP报文的头部加上表示数据长度。
4. 应用层发送数据时**定长**发送。