一、基础篇

1、ISO七层网络模型，各层的作用?

> 七层网络模型自下往上主要包括：物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。
>
> - 物理层：主要作用是**实现相邻节点间比特流的透明传输，尽可能避免传输介质和物理设备的差异。**。具体做法是在发送端将1 、0 转化为电流强弱来进行传输，在到达目的地后再将电流强弱转化为0、1 ， 也就是我们常说的**模数转换与数模转换**。这一层的数据叫作**比特。**
> - 数据链路层：主要用于**对数据包中的MAC 地址进行解析和封装。**这一层的数据叫作**数据帧。**在这一层工作的设备是网卡、网桥、交换机。
> - 网络层：主要用于**对数据包中的 ip地址进行封装和解析**，这一层的数据叫作**数据包。**在这一层工作的设备有路由器、交换机等。
> - 传输层：定义了传输数据的协议和端口号，主要**用于数据的分段、传输和重组。**在这一层工作的协议有TCP 和UDP 等。 TCP 是传输控制协议，传输效率低，可靠性强，用于传输对可靠性要求高、数据量大的数据，比如支付宝转账使用的就是 TCP；UDP 是用户数据报协议，与TCP 的特性恰恰相反，用于传输可靠性要求不高、数据量小的数据，例如抖音等视频服务就使用了UDP 。
> - 会话层：在传输层的基础上**建立连接和管理会话**，具体包括登录验证、断点续传、数据粘包与分包等。在设备之间需要互相识别的可以是IP， 也可以是MAC 或者主机名。
> - 表示层：主要**对接收的数据进行解释、加密、解密、压缩、解压缩等**，即把计算机能够识别的内容转换成人能够识别的内容（图片、声音、文字等）。
> - 应用层：**基于网络构建具体应用**，例如FTP 文件上传下载服务、Telnet 服务、HTTP 服务、DNS 服务、SNMP 邮件服务等。

2、浏览器输入网址到网页显示，期间发生了什么?

> 1、浏览器解析URL，生成web服务器的请求信息，生成HTTP请求报文。
>
> 2、**进行DNS解析**，根据域名查找对应的ip地址。首先查找浏览器自身的缓存中有没有对应域名的ip地址，如果有直接返回，如果没有则循环操作系统，如果还是没有则在hosts文件中查找，如果还是没有则发送一个DNS请求循环本地域名服务器，如果本地域名服务器还是没有，本地域名服务器会去问它的根域名服务器，根域名服务器不直接用于域名解析而是告诉请求方取指定的顶级域名服务器的地址，让请求方去访问这个顶级域名的地址，顶级域名也是不负责映射域名，而是告诉请求方去指定的权威域名服务器中查找，最后请求方会到指定的权威域名服务器中查找，找到之后则返回给本地域名服务器。然后逐层反向回去，一直到浏览器，且在本地域名服务器、操作系统、浏览器缓存中都会保存找到的这个域名映射关系。
>
> 3、进行**三次握手建立tcp连接。**
>
> 4、将**http请求报文发送给服务器。**
>
> 5、服务器解析请求报文，**响应对应的数据。**
>
> 6、浏览器接收到数据之后在**浏览器中渲染。**
>
> 7、断开浏览器和服务器之间的tcp连接。

二、HTTP篇

1、HTTP是什么？说说HTTP？

> HTTP 是**超文本传输协议**。是一个在计算机世界里专门在**「两点」**之间**「传输」**文字、图片、音频、视频等**「超文本」**数据的
> **「约定和规范」**。

2、HTTP常见的状态码有哪些？

1xx	**提示信息，表示目前处于协议处理的中间状态，还有后续操作。比如：POST产生两个TCP数据包。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。**
2xx	成功状态码，报文已经收到且被正确处理。 「**200 OK**」是最常见的成功状态码，表示一切正常。 「**204 No Content**」也是常见的成功状态码，与 200 OK 基本相同，但响应头**没有 body 数据**。 「**206 Partial Content**」是**应用于 HTTP 分块下载或断点续传**，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理成功的状态。
3xx	表示客户端请求的资源发生了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是**重定向**。 「**301 Moved Permanently**」表示**永久重定向**，说明请求的资源已经不存在了，需改用新的 URL 再次访问。 「**302 Found**」表示**临时重定向**，说明请求的资源还在，但暂时需要用另一个 URL 来访问。 「**304 Not Modified**」不具有跳转的含义，表示资源未修改，**重定向已存在的缓冲文件**，也称缓存重定向，也就是告诉客户端可以继续使用缓存资源，用于缓存控制。
4xx	状态码表示客户端发送的**报文有误**，服务器无法处理，也就是错误码的含义。 「**400 Bad Request**」表示客户端请求的报文有错误，但只是个笼统的错误。 「**403 Forbidden**」表示服务器**禁止访问资源**，并不是客户端的请求出错。 「**404 Not Found**」表示请求的资源在服务器上**不存在或未找到**，所以无法提供给客户端。
5xx	表示客户端请求报文正确，但是**服务器处理时内部发生了错误**，属于服务器端的错误码。 「**500 Internal Server Error**」与 400 类型，是个笼统通用的错误码，服务器发生了什么错误，我们并不知道。 「**501 Not Implemented**」表示客户端请求的功能还不支持，类似“即将开业，敬请期待”的意思。 「**502 Bad Gateway**」通常是服务器**作为网关或代理时返回的错误码，表示服务器自身工作正常**，访问后端服务器发生了错误。 「**503 Service Unavailable**」表示**服务器当前很忙，暂时无法响应客户端**，类似“网络服务正忙，请稍后重试”的意思。

3、HTTP常见字段有哪些？

```
Host  # 客户端发送请求时，用来指定服务器的域名。比如Host:www.baidu.com
Content-Length  # 服务器在返回数据时，会有 Content-Length 字段，表明本次回应的数据长度。可以解决粘包问题。
Connection   # 最常用于客户端要求服务器使用「HTTP 长连接」机制，以便其他请求复用。
Connection: Keep-Alive
Content-Type   # 用于服务器回应时，告诉客户端，本次数据是什么格式的，html文件，图片或者音频等
Content-Encoding  # 字段说明数据的压缩方法。表示服务器返回的数据使用了什么压缩格式。客户但在请求时Accept-Encoding告知接受压缩格式
```

4、GET和POST的区别?

> 根据规范，**GET 的语义是从服务器获取指定的资源**，这个资源可以是静态的文本、页面、图片视频等。GET 请求的参数位置一般是写在 URL 中，URL 规定只能支持 ASCII，所以 GET 请求的参数只允许 ASCII 字符 ，而且浏览器会对 URL 的长度有限制（HTTP协议本身对 URL长度并没有做任何规定）。GET的请求记录会被浏览器所记录，其请求的url会被完整记录在历史记录中。
>
> 根据规范，**POST 的语义是根据请求报文对指定的资源做出处理**，具体的处理方式视资源类型而不同。POST 请求携带数据的位置一般是写在报文 body 中，body 中的数据可以是任意格式的数据，只要客户端与服务端协商好即可，而且浏览器不会对 body 大小做限制。POST请求是不会被浏览器缓存的，无法在历史记录中找到。
>
> 根据规范，GET是幂等和安全的，而POST则不是。
>
> 在 HTTP 协议里，所谓的**「安全」是指请求方法不会「破坏」服务器上的资源。**
> 所谓的**「幂等」，意思是多次执行相同的操作，结果都是「相同」的。**

5、GET和POST是安全和幂等的吗?

> **GET 方法就是安全且幂等的**，因为它是「只读」操作，无论操作多少次，服务器上的数据都是安全的，且每次的结果都是相同的。所以，**可以对 GET 请求的数据做缓存，这个缓存可以做到浏览器本身上（彻底避免浏览器发请求），也可以做到代理上（如nginx），而且在浏览器中 GET 请求可以保存为书签。**
>
> **POST** 因为是「新增或提交数据」的操作，会修改服务器上的资源，所以是**不安全**的，且多次提交数据就会创建多个资源，所以**不是幂等**的。所以，**浏览器一般不会缓存 POST 请求，也不能把 POST 请求保存为书签。**

6、HTTP缓存技术?

HTTP缓存技术可以避免每次请求都从服务器中获取，直接取本地缓存的数据即可。

HTTP缓存有两种实现方式：**强制缓存**和**协商缓存。**

> 强制缓存：只要浏览器判断缓存没有过期，则直接使用浏览器的本地缓存，决定是否使用缓存的主动性在于浏览器这边。
>
> 主要通过Cache-Control（一个相对时间）和Expires（绝对时间）两个响应头部字段实现，都用来表示资源在客户端缓存的有效期。如果同时出现这两个响应头，Cache-Control优先于Expires使用。具体实现流程如下：
> 1、当浏览器第一次请求访问服务器资源时，服务器会在返回这个资源的同时，在 Response 头部加上 Cache-Control，Cache-Control 中设置了过期时间大小；
> 2、浏览器再次请求访问服务器中的该资源时，会先**通过请求资源的时间与 Cache-Control 中设**

> **置的过期时间大小，来计算出该资源是否过期**，如果没有，则使用该缓存，否则重新请求服务器；
> 3、服务器再次收到请求后，会再次更新 Response 头部的 Cache-Control。

> 协商缓存：通过服务端告知客户端是否可以使用缓存的方式被称为协商缓存。比如304响应码，就是告诉浏览器可以使用本地缓存的资源。
>
> 协商缓存基于两种头部实现。
>
> 第一种：请求头部中的 `If-Modified-Since` 字段与响应头部中的 `Last-Modified` 字段实现。
> Last-Modified 是服务器响应头中的字段，它表示资源的最后修改时间。当浏览器再次请求该资源时，会携带 If-Modified-Since 字段，将资源的最后修改时间发送给服务器。如果服务器判断该资源的最后修改时间与 If-Modified-Since 字段相同，则返回 304 Not Modified 状态码，告诉浏览器可以使用缓存的副本。
>
> 第二种：请求头部中的 `If-None-Match` 字段与响应头部中的 `ETag` 字段。
> ETag 是服务器响应头中的字段，它是一个唯一标识符，表示资源的版本号。当浏览器再次请求该资源时，会携带 If-None-Match 字段，将资源的 ETag 值发送给服务器。如果服务器判断该资源的 ETag 值与 If-None-Match 字段相同，则返回 304 Not Modified 状态码，告诉浏览器可以使用缓存的副本。
>
> 第一种实现方式是基于时间实现的，第二种实现方式是基于一个唯一标识实现的，相对来说后者可以更加准确地判断文件内容是否被修改，避免由于时间篡改导致的不可靠问题。
>
> **ETag方式优先级高于Last-Modified**，因为ETag主要能解决Last-Modified无法解决的几个问题：
> 1、在没有修改文件内容情况下文件的最后修改时间可能也会改变，这会导致客户端认为这文件被改动了，从而重新请求；
> 2、可能有些文件是在秒级以内修改的，`If-Modified-Since` 能检查到的粒度是秒级的，使用Etag就能够保证这种需求下客户端在 1 秒内能刷新多次；
> 3、有些服务器不能精确获取文件的最后修改时间。

7、HTTP/1.1的优点？缺点？

> HTTP发展：HTTP/1.0 --> HTTP/1.1 --> HTTPS --> HTTP/2.0 --> HTTP/3.0
>
> HTTP 最突出的优点是「**简单、灵活和易于扩展、应用广泛和跨平台**」。
>
> **HTTP/1.1优点：**
> 1、**简单。**HTTP的基本报文格式就是`header + body`，头部信息使用key-value简单文本的形式，易于理解，降低了学习和使用的门槛。
> 2、**灵活和易用扩展。**HTTP 协议里的各类请求方法、URI/URL、状态码、头字段等每个组成要求都没有被固定死，都允许开发人员**自定义和扩充。**同时 HTTP 由于是工作在应用层（`OSI` 第七层），则它**下层可以随意变化**，比如：1、HTTPS 就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层；2、HTTP/1.1 和 HTTP/2.0 传输协议使用的是 TCP 协议，而到了 HTTP/3.0 传输协议改用了 UDP 协议。
> 3、**应用广泛和跨平台。**无论是客户端浏览器还是客户端软件都可以使用http协议。
>
> **HTTP/1.1缺点：**
> 1、无状态：服务器没有记忆能力，他在完成关联性的操作时会非常麻烦。需要借助cookie技术，每次请求都要将cookie发送给服务器以便验证身份等信息。（无状态也有好处，服务器不需要记录额外的资源来记录状态信息，减轻了服务器的负担，更好地对外提供服务）。
> 2、明文传输：在传输过程中，明文传输相当于信息裸奔，信息内容没有隐私可言，容易被窃取。（方便阅读，给调试工作带来了极大便利）。
> 3、不安全：通信使用明文、不验证通信方的身份、无法证明报文的完整性。
> HTTP的安全问题，目前使用HTTPS进行解决，在HTTP和TCP层之间引入了SSL/TLS层。

8、HTTP/1.1的性能如何?

> HTTP 协议是基于 **TCP/IP**，并且使用了「**请求 - 应答**」的通信模式，所以性能的关键就在这**两点**里。
>
> 1、**长连接**：早期HTTP/1.0使用短连接，每次发送一个请求，都要新建一个TCP连接，而且是串行请求，做了很多无谓的TCP连接建立和断开操作，增加了通信开销。HTTP/1.1 提出了**长连接**的通信方式，也叫持久连接。这种方式的好处在于减少了 TCP 连接的重复建立和断开所造成的额外开销，减轻了服务器端的负载。持久连接的特点是，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。（有超时设置）
> 2、**管道网络传输**：可**在同一个 TCP 连接里面，客户端可以发起多个请求**，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以**减少整体的响应时间。**但是，**服务器必须按照接受请求的顺序发送对这些管道化请求的响应。**
> **对头堵塞**：如果服务器在处理A请求时比较长，那么后续的请求在客户端无法被发送。HTTP/1.0会有这种问题，HTTP/1.1解决了对头堵塞的问题，但是没有解决响应的对头阻塞。
> 3、**对头阻塞**：响应时的阻塞问题。
>
> 总之：HTTP/1.1的性能一般般，后续的HTTP/2.0和HTTP/3就是在优化HTTP的性能。

9、HTTP和HTTPS有什么区别?

> - HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题。HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。
> - HTTP 连接建立相对简单， TCP 三次握手之后便可进行 HTTP 的报文传输。而 HTTPS 在 TCP 三次握手之后，还需进行 SSL/TLS 的握手过程，才可进入加密报文传输。
> - 两者的默认端口不一样，HTTP 默认端口号是 80，HTTPS 默认端口号是 443。
> - HTTPS 协议需要向 CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的。

10、HTTPS解决了HTTP的哪些问题?

> HTTP是明文传输的，所以在安全上存在三个风险：窃听风险、篡改风险和冒充风险。
>
> HTTP**S** 在 HTTP 与 TCP 层之间加入了 `SSL/TLS` 协议，可以很好的解决了上述的风险：
>
> - **信息加密**：交互信息无法被窃取。
> - **校验机制**：无法篡改通信内容，篡改了就不能正常显示，但百度「竞价排名」依然可以搜索垃圾广告。
> - **身份证书**：证明淘宝是真的淘宝网。

18、HTTP/1.1相比于HTTP/1.0提高了什么性能?

> - 使用长连接的方式改善了 HTTP/1.0 短连接造成的性能开销。
> - 支持管道（pipeline）网络传输，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以减少整体的响应时间。
>
> 但 HTTP/1.1 还是有性能瓶颈：
>
> - 请求 / 响应头部（Header）未经压缩就发送，首部信息越多延迟越大。只能压缩 `Body` 的部分；
> - 发送冗长的首部。每次互相发送相同的首部造成的浪费较多；

- 服务器是按请求的顺序响应的，如果服务器响应慢，会招致客户端一直请求不到数据，也就是响应的队头阻塞；
- 没有请求优先级控制；
- 请求只能从客户端开始，服务器只能被动响应。

19、HTTP/2做了什么优化?

HTTP/2 协议是基于 HTTPS 的，所以 HTTP/2 的安全性也是有保障的。
HTTP/2主要在**头部压缩、二进制格式、并发传输和服务器主动推送资源**方面做了优化。

1、头部压缩
HTTP/2 会**压缩头**（Header）如果你同时发出多个请求，他们的头是一样的或是相似的，那么，协议会帮你**消除重复的部分。**这就是所谓的 `HPACK` 算法：在客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就**提高速度**了。
2、二进制格式
HTTP/2 不再像 HTTP/1.1 里的纯文本形式的报文，而是全面采用了**二进制格式**，头信息和数据体都是二进制，并且统称为帧（frame）：**头信息帧（Headers Frame）和数据帧（Data Frame）**。这加大增加了数据传输效率。
3、并发传输
引出了 Stream 概念，多个 Stream 复用在一条 TCP 连接。1 个 TCP 连接包含多个 Stream，Stream 里可以包含 1 个或多个 Message，Message 对应 HTTP/1 中的请求或响应，由 HTTP 头部和包体构成。Message 里包含一条或者多个 Frame，Frame 是 HTTP/2 最小单位，以二进制压缩格式存放 HTTP/1 中的内容（头部和包体）。**针对不同的 HTTP 请求用独一无二的 Stream ID 来区分，接收端可以通过 Stream ID 有序组装成 HTTP 消息，不同 Stream 的帧是可以乱序发送的，因此可以并发不同的 Stream ，也就是 HTTP/2 可以并行交错地发送请求和响应。**
4、服务器推送
客户端和服务器**双方都可以建立 Stream**， Stream ID 也是有区别的，客户端建立的 Stream 必须是奇数号，而服务器建立的 Stream 必须是偶数号。比如，客户端通过 HTTP/1.1 请求从服务器那获取到了 HTML 文件，而 HTML 可能还需要依赖 CSS 来渲染页面，这时客户端还要再发起获取 CSS 文件的请求，需要两次消息往返。如果是HTTP/2，则可以直接将css文件推送到客户端，而不需要客户端主动发出请求。

20、HTTP/2有什么缺陷?

HTTP/2 通过 Stream 的并发能力，解决了 HTTP/1 响应时的队头阻塞的问题，看似很完美了，但是 HTTP/2 还是存在"队头阻塞"的问题，只不过问题不是在 HTTP 这一层面，而是在 TCP 这一层。

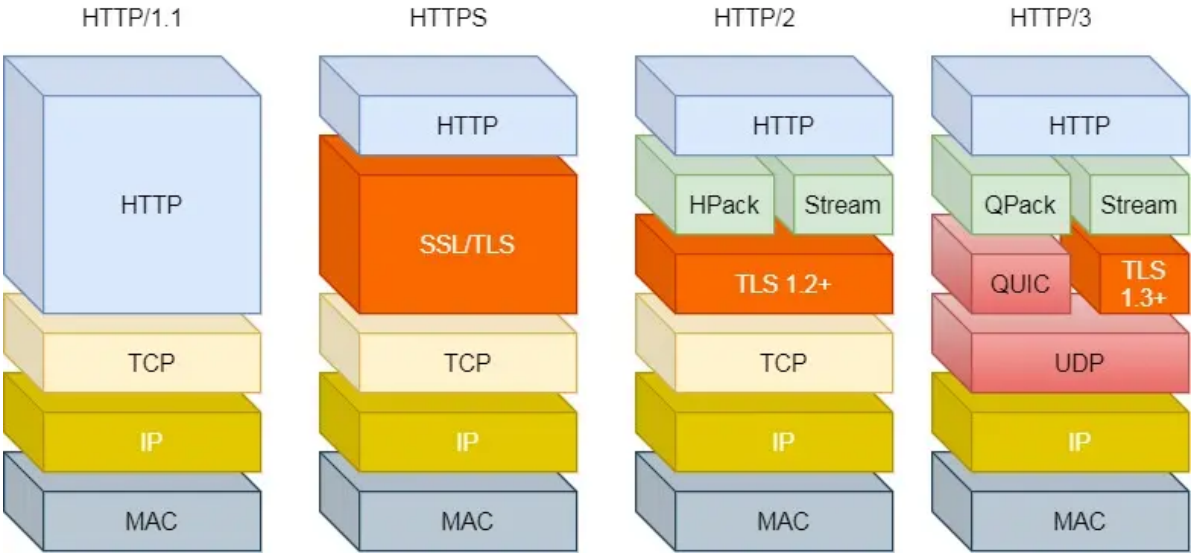
HTTP/2 是基于 TCP 协议来传输数据的，TCP 是字节流协议，TCP 层必须保证收到的字节数据是完整且连续的，这样内核才会将缓冲区里的数据返回给 HTTP 应用，那么当「前 1 个字节数据」没有到达时，后收到的字节数据只能存放在内核缓冲区里，只有等到这 1 个字节数据到达时，HTTP/2 应用层才能从内核中拿到数据，这就是 HTTP/2 队头阻塞问题。

所以，一旦发生了丢包现象，就会触发 TCP 的重传机制，这样在一个 TCP 连接中的**所有的 HTTP 请求都必须等待这个丢了的包被重传回来。**

21、HTTP/3做什么什么优化?

- HTTP/1.1 中的管道（ pipeline）虽然解决了请求的队头阻塞，但是**没有解决响应的队头阻塞**，因为服务端需要按顺序响应收到的请求，如果服务端处理某个请求消耗的时间比较长，那么只能等响应完这个请求后， 才能处理下一个请求，这属于 HTTP 层队头阻塞。

> - HTTP/2 虽然通过多个请求复用一个 TCP 连接解决了 HTTP 的队头阻塞 ，但是**一旦发生丢包，就会阻塞住所有的 HTTP 请求**，这属于 TCP 层队头阻塞。
>
> HTTP/2 队头阻塞的问题是因为 TCP，所以 **HTTP/3 把 HTTP 下层的 TCP 协议改成了 UDP！**



UDP 发送是不管顺序，也不管丢包的，所以不会出现像 HTTP/2 队头阻塞的问题。大家都知道 UDP 是不可靠传输的，但基于 UDP 的 **QUIC 协议** 可以实现类似 TCP 的可靠性传输。

> QUIC有三个特点：
>
> 1、无对头阻塞
> QUIC 协议也有类似 HTTP/2 Stream 与多路复用的概念，也是可以在同一条连接上并发传输多个 Stream，Stream 可以认为就是一条 HTTP 请求。**当某个流发生丢包时，只会阻塞这个流，其他流不会受到影响，因此不存在队头阻塞问题。**这与 HTTP/2 不同，HTTP/2 只要某个流中的数据包丢失了，其他流也会因此受影响。所以，QUIC 连接上的多个 Stream 之间并没有依赖，都是独立的，某个流发生丢包了，只会影响该流，其他流不受影响。
>
> 2、更快的连接建立
> HTTP/3 在传输数据前虽然需要 QUIC 协议握手，但是这个握手过程只需要 1 RTT，握手的目的是为确认双方的「连接 ID」，连接迁移就是基于连接 ID 实现的。
>
> 3、连接迁移
> 基于 TCP 传输协议的 HTTP 协议，由于是通过四元组（源 IP、源端口、目的 IP、目的端口）确定一条 TCP 连接。 QUIC 是一个在 UDP 之上的**伪** TCP + TLS + HTTP/2 的多路复用的协议。
> **当移动设备的网络从 4G 切换到 WIFI 时，意味着 IP 地址变化了，那么就必须要断开连接，然后重新建立连接。**而建立连接的过程包含 TCP 三次握手和 TLS 四次握手的时延，以及 TCP 慢启动的减速过程，给用户的感觉就是网络突然卡顿了一下，因此连接的迁移成本是很高的。而 QUIC 协议没有用四元组的方式来“绑定”连接，而是通过**连接 ID** 来标记通信的两个端点，客户端和服务器可以各自选择一组 ID 来标记自己，因此即使移动设备的网络变化后，导致 IP 地址变化了，只要仍保有上下文信息（比如连接 ID、TLS 密钥等），就可以“无缝”地复用原连接，消除重连的成本，没有丝毫卡顿感，达到了**连接迁移**的功能。

三、其他

2、说一下TCP的三次握手?

> 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字段中包含标志位SYN=1，序列号seq=x。第一次握手前客户端的状态是close状态的，第一次握手之后客户端状态变为SYN_SEND。此时服务端的状态为LISTEN监听状态。
>
> 第二次握手：服务器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，然后客户端回复一段报文，其中包括了标志位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监听状态，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_SEND。（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：客户端收到服务器发来的报文后，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，第三次握手后，客户端和服务端的状态都会变为ESTABLISHED。此时连接建立完成。
>
> 三次握手交换了：初始序列号ISN、最大段大小MSS、窗口大小Win、窗口缩放因子WS、是否支持选择确认SACK_PERM。

3、两次握手可以吗?

> 之所以需要第三次握手，主要是**为了防止已经失效的连接请求报文段突然又传输到了服务器**，从而导致服务器再次建立与客户端的异常连接。此外，也**无法可靠的同步双方初始序列号。**
>
> 比如：一个客户端向服务器发起了一个连接请求的报文段，但是这个报文段在网络出出现了极大的延迟。因为一直没有收到确认信息，客户端又发出了一个请求连接的报文段，并且成功建立了连接，之后交互完成后双方都释放了连接。而在此之后，之前在网络中延迟的报文段发送到了服务器，如果是两次握手，那么服务器会以为有新的连接请求，发出确认报文段之后就建立了连接。此时客户端不会响应服务器的确认且不发送数据，而服务器会一直等待客户端发送数据，从而造成资源浪费。
>
> 四次握手：三次握手就已经理论上最少可靠建立连接，所以不需要使用更多的通信次数，多了也是浪费。

4、说一下四次挥手?

> 第一次挥手：客户端应用进程调用断开连接的请求，向服务器发送一个带有终止标志位FIN=1，序列号seq=u的报文段。并停止再发送数据，主动关闭TCP连接，进入FIN-WAIT-1状态，等待服务器确认。

第二次挥手：服务器收到连接断开的报文段后即发出确认报文段（ACK=1, seq=v, ack=u+1），进入CLOSE-WAIT半关闭状态。客户端收到服务器的确认后，进入FIN-WAIT-2状态，等待服务器发出的连接释放报文段。

第三次挥手：服务器发送完数据后，就会发出连接释放报文段（FIN=1、ACK=1、seq=w、ack=u+1），服务器进入LAST-ACK最后确认状态，等待客户端的确认。

第四次挥手：客户端收到服务器释放连接的报文段后，对此发出确认报文段（ACK=1、seq=u+1、ack=w+1），此时客户端进入TIME-WAIT等待状态，此时需要经过2个最大报文段生存时间后，客户端才进入关闭状态。服务器收到客户端发出的确认报文段后关闭连接，若没有收到客户端的确认报文段后，服务器就会重传连接释放报文段。

5、第四次挥手为什么要等待2MSL?

1、**保证客户端发送的最后一个ACK报文段能够到达服务器。**这个`ACK`报文段有可能丢失，服务器收不到这个确认报文，就会超时重传连接释放报文段，然后客户端可以在`2MSL`时间内收到这个重传的连接释放报文段，接着客户端重传一次确认，重新启动2MSL计时器，最后客户端和服务器都进入到`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，则无法收到服务器重传的连接释放报文段，所以不会再发送一次确认报文段，服务器就无法正常进入到`CLOSED`状态。

2、**防止已失效的连接请求报文段出现在本连接中。**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。

6、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。

7、说说TCP报文首部有哪些字段，其作用是什么?

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位序列号seq：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ack：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。

8、TCP有哪些特点?

TCP是面向连接的传输层协议。

它是端到端的，每一条TCP连接只能有两个端点。

TCP通过流量控制、拥塞控制等方式实现可靠交付的服务。

TCP提供了全双工通信，双方既可以接收也可以发送数据。

TCP是面向字节流的数据传输。

9、TCP和UDP的区别?

1、TCP是面向连接的；UDP是无连接的，即发送数据之前不需要建立连接。

2、TCP提供流量控制、拥塞控制等方式提供可靠的交付服务；UDP不保证可靠的交付服务，数据丢失也不会重传。

3、TCP面向字节流，把数据看成一串无结构的字节流；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。

5、每一条TCP连接只是端到端的；UDP支持一对多、一对一、多对多的通信方式。

6、TCP首部字节开销是20个字节；UDP首部开销是8个字节。

10、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。

11、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。
>
> 注意：UDP没有粘包与拆包的问题，因为UDP是面向用户数据报的数据传输，有数据边界。
>
> 为什么会产生粘包和拆包?
> 1、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;
> 2、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;
> 3、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;
> 4、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:
> 1、发送端将每个数据包封装为固定长度;
> 2、在数据尾部增加特殊字符进行分割;
> 3、将数据分为两个部分，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。

12、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，断开连接基于四次挥手。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，并且保证数据包按序达到。
>
> TCP的可靠性还体现在可控制。他有数据包校验、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、流量控制和拥塞控制等机制。

13、说一下TCP的滑动窗口机制?

> TCP利用滑动窗口实现流量控制。流量控制是为了控制发送方速率，保证接收方来得及接收。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（在三次握手时就通告了客户端接收窗口的大小）。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。
>
> TCP头包含window字段，16bit位，它代表的是窗口的字节容量，最大为65535。这个字段是接收端告诉发送端自己还有多少缓冲区可以接收数据。于是发送端就可以根据这个接收端的处理能力来发送数据，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。

14、讲一下拥塞控制?

> 防止过多的数据注入到网络中，从而造成过严重的网络拥塞。几种拥塞控制方法：慢启动、拥塞避免、快速重传和快速恢复。
>
> **1、慢启动**
>
> 把拥塞窗口cwnd设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至少一个MSS的数值。每经过一个传输轮次，拥塞窗口cwnd就加倍。为了防止拥塞窗口cwnd增加过大引起网络拥塞，还需要设置一个慢启动门限ssthresh状态变量。

当cwnd < ssthresh时，使用慢启动算法。
当cwnd > ssthresh时，停止使用慢启动算法而改用拥塞避免算法。
当cwnd = ssthresh时，既可以使用慢启动算法，也可以使用拥塞避免算法。

2、拥塞避免

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cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长。

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（其根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络中的分组数，使得发生 拥塞的路由器有足够时间把队列中积压的分组处理完毕。

3、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。此时拥塞窗口cwnd = cwnd / 2 设置为原来的一半，ssthresh设置为拥塞窗口大小cwnd。

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%。

4、**快恢复**

快速重传和快速恢复算法一般同时使用，快速恢复算法是认为，你还能收到 3 个重复 ACK 说明网络也不那么糟糕，所以没有必要像 `RTO` 超时那么强烈。

算法：

- 拥塞窗口 `cwnd = ssthresh + 3` （ 3 的意思是确认有 3 个数据包被收到了，cwnd和ssthredh是被快重传更新过的）；
- 重传丢失的数据包；
- 如果再收到重复的 ACK，那么 cwnd 增加 1；
- 如果收到新数据的 ACK 后，把 cwnd 设置为第一步中的 ssthresh 的值，原因是该 ACK 确认了新的数据，说明从 duplicated ACK 时的数据都已收到，该恢复过程已经结束，可以回到恢复之前的状态了，也即再次进入拥塞避免状态。

15、说一下TCP的重传机制?

TCP的重传机制主要有超时重传、快速重传、SAKCK和D-SACK。所有TCP指针数据包丢失的情况，都会使用重传机制解决。

（1）超时重传

超时重传就是在发送数据时，设定一个定时器，当超过指定的时间后，没有收到对方的ACK确认应答报文，就会重发该数据。
TCP在**数据包丢失**和**确认应答丢失**的情况下都会出发超时重传。RTT指的是数据发送时刻到接收到确认的时刻的差值，也就是包的往返时间。超时重传时间RTO的值应该略大于RTT的值，并且是动态变换的。

（2）快速重传

快速重传机制不以时间为驱动，而是以数据驱动重传。快速重传的工作方式是当收到三个相同的ACK报文时，会在定时器过期之前，重传丢失的报文段。快速重传只解决了超时时间的问题，但是还面临着另一个问题，也就是重传的时候，是重传一个，还是重传所有的。为了解决这个问题，引入了SACK方法。

（3）SACK选择性确认

这种方式需要在 TCP 头部「选项」字段里加一个 `SACK` 的东西，它**可以将已收到的数据的信息发送给「发送方」**，这样发送方就可以知道哪些数据收到了，哪些数据没收到，知道了这些信息，就可以**只重传丢失的数据。**

（4）Duplicate SACK

其主要**使用了 SACK 来告诉「发送方」有哪些数据被重复接收了。**

1. 可以让「发送方」知道，是发出去的包丢了，还是接收方回应的 ACK 包丢了;
2. 可以知道是不是「发送方」的数据包被网络延迟了;
3. 可以知道网络中是不是把「发送方」的数据包给复制了;

16、说一下滑动窗口和流量控制?

一、滑动窗口

TCP是每发送一条数据，都要进行一次确认应答。当上一个数据包收到了应答了，再发送下一个。这种传输方式会有一个缺点，也就是数据包的往返时间越长，通信的效率就越低，网络的吞吐量就会越低。为了解决这个问题引入了窗口的概念。

引入滑动窗口之后，就可以指定窗口大小，窗口大小就是指无需等待确认应答，而可以继续发送数据的最大值。窗口的实现实际上是操作系统开辟的一个缓存空间，发送方主机在等到确认应答返回之前，必须在缓冲区中保留已发送的数据。如果按期收到确认应答，此时数据就可以从缓存区清楚。

TCP头中有一个Window字段，也就是窗口的大小。这个字段是告诉服务器告诉客户端自己还有多少缓冲区可以接收数据。于是客户端就可以根据这个服务器的处理能力来发送数据，而不会导致接收端处理不过来。所以**窗口的大小由服务器窗口大小来决定。**如果客户端发送的数据不能超过服务器的窗口大小，否则服务器就无法正常接收数据了。

二、拥塞控制和流量控制的区别?

拥塞控制：拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况；常用的方法就是：（1）慢开始、拥塞避免（2）快重传、快恢复。

流量控制：流量控制是作用于接收者的，它是控制发送者的发送速度从而使接收者来得及接收，防止分组丢失的。

17、ARP地址解析协议的工作过程?

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。该协议属于数据链路层。

- 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。

18、什么是Dos、DDos、DRDos攻击?

DoS、DDoS和DRDoS攻击都是网络安全领域中的攻击类型，它们的目标都是使目标系统无法正常工作或服务不可用。

1. **DoS 攻击（Denial of Service 攻击）：** DoS攻击旨在通过发送大量的请求或恶意数据流来超载目标系统，使其无法响应合法用户的请求。这种攻击会耗尽目标系统的资源（如带宽、处理能力、内存等），导致系统崩溃或变得不稳定。DoS攻击通常是由单个攻击者发起，目的是削弱系统的性能或使其无法正常运行。最常见的DoS攻击就有**计算机网络宽带攻击**、**连通性攻击**。
2. **DDoS 攻击（Distributed Denial of Service 攻击）：** DDoS攻击是DoS攻击的升级版本，它不仅涉及一个攻击者，而是由多个分布在不同地理位置的攻击者共同发起。这些攻击者通常控制了一个或多个被感染的计算机，形成了一个“僵尸网络”（botnet）。通过协调这些被感染的计算机，攻击者可以同时发起大规模的攻击，更加有效地使目标系统不可用。常见的DDos有**SYN Flood、Ping of Death、ACK Flood、UDP Flood**等。
3. **DRDoS 攻击（Distributed Reflection Denial of Service 攻击）：** DRDoS攻击是DDoS攻击的一种变种，它利用了一些网络协议的特性，例如UDP（用户数据报协议）的源IP地址欺骗。攻击者向一个或多个服务器发送伪造的请求，将响应重定向到目标系统的IP地址。这样，攻击者可以利用服务器的响应来放大攻击流量，从而使目标系统的带宽和资源负载急剧增加，导致服务不可用。

总之，DoS攻击旨在通过超载目标系统来拒绝服务，DDoS攻击通过多个攻击者和分布式的方式来放大攻击效果，而DRDoS攻击则是一种基于反射的DDoS攻击，利用了某些网络协议的特性来放大攻击流量。这些攻击类型对于网络和系统的稳定性和可用性构成了严重威胁，防御和缓解这些攻击需要综合的网络安全策略和工具。

19、说说防盗链?

盗链是指服务提供商自己不提供服务的内容，通过技术手段（可以理解成爬虫）去获取其他网站的资源展示在自己的网站上。常见的盗链有以下几种：图片盗链、音频盗链、视频盗链等。

网站盗链会大量消耗被盗链网站的带宽，而真正的点击率也许会很小，严重损害了被盗链网站的利益。

被盗网站就自然会**防盗链**，可以通过经常更换图片名，也可以通过检测referer。因为正常用户访问一张图片一定是从自己的网站点击链接进去的，如果一个请求的referer是其他网站，就说明这是一个爬虫。

20、ICMP协议的功能是是什么?

> 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。

21、ping的原理是什么?

> ping，**Packet Internet Groper**，是一种因特网包探索器，用于测试网络连接量的程序。Ping是工作在TCP/IP网络体系结构中应用层的一个服务命令， 主要是向特定的目的主机发送ICMP（Internet Control Message Protocol 因特网报文控制协议） 请求报文，测试目的站是否可达及了解其有关状态。
>
> 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

22、TCP如何保证可靠传输?

> 可靠传输就是通过TCP连接传送的数据是没有差错、不会丢失、不重复并且按序到达的。TCP是通过序列号、检验和、确认应答信号、重发机制、连接管理、窗口控制、流量控制、拥塞控制一起保证TCP传输的可靠性的。
>
> 可靠传输的具体实现是： 应用层的数据会被分割成TCP认为最适合发送的数据块。
>
> 1、序列号：TCP给发送的每一个包都进行编号，接收方对数据包进行排序，把有序数据传送给应用层，TCP的接收端会丢弃重复的数据。
>
> 2、检验和：TCP将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。
>
> 3、确认应答：如果收到的数据报报文段的检验和没有差错，就确认收到，如果有差错，TCP就丢弃这个报文段和不确认收到此报文段。
>
> 4、流量控制：TCP 连接的每一方都有固定大小的缓冲空间，TCP的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议。
>
> 5、拥塞控制：当网络拥塞时，减少数据的发送。

6、停止等待协议：它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组。

7、超时重传： 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。

1、HTTP协议的特点?

HTTP超文本传输协议通常是一种用于在Web上进行数据传输的协议。具备以下特点：

1. **无连接性（Connectionless）：** HTTP是一种无连接的协议，这意味着每次请求都是独立的，服务器不会保存与之前请求的任何关联信息。这样的设计使得服务器可以更快地释放资源，但也需要在每次请求和响应中包含足够的信息来确保正确的数据传输。
2. **无状态性（Stateless）：** HTTP是无状态的，服务器不会保持任何关于客户端的状态信息。每个请求都是独立的，服务器不会记住之前的请求或会话状态。为了处理用户状态，需要使用额外的机制，如Cookies和Session。
3. **基于请求和响应（Request-Response）模型：** HTTP采用了请求和响应的交互模式。客户端发送请求到服务器，服务器处理请求并返回相应的响应。这种模型使得客户端能够明确地发起请求并等待响应，从而实现了有效的通信。
4. **支持多媒体：** HTTP支持传输不同类型的媒体数据，如文本、图像、音频和视频等。这种灵活性使得它成为了在Web上传输多种内容的理想协议。
5. **简单和可扩展：** HTTP的设计简单且灵活，易于理解和实现。它采用了一些基本的方法和状态码，同时也允许通过HTTP头部来添加自定义的信息。这使得HTTP可以根据需要进行扩展，适应不同的应用场景。
6. **支持缓存：** HTTP支持缓存机制，可以在客户端或代理服务器上保存已获取的资源副本，从而减少重复请求，提高性能和效率。
7. **基于文本和可读性：** HTTP的请求和响应是以文本形式进行传输的，这使得开发人员和管理员可以轻松地监测和调试通信。然而，这也可能导致数据传输效率较低，但可以通过压缩等技术来缓解这个问题。

2、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。

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。

HTTP 状态码分为 5 大类：1XX：表示消息状态码；2XX：表示成功状态码；3XX：表示重定向状态码；4XX：表示客户端错误状态码；5XX：表示服务端错误状态码。其中常见的具体状态码有：200：请求成功；301：永久重定向；302：临时重定向；404：无法找到此页面；405：请求的方法类型不支持；500：服务器内部出错。

3、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。

4、POST和GET的区别?

> GET 和 POST 最本质的区别是规范上的区别，在规范中，定义 GET 请求是用来获取资源的，也就是进行查询操作的，而 POST 请求是用来传输实体对象的，因此会使用 POST 来进行添加、修改和删除等操作。
>
> GET请求参数通过URL传递，POST的参数放在请求体中。
>
> GET 请求可以直接进行回退和刷新，不会对用户和程序产生任何影响；而 POST 请求如果直接回滚和刷新将会把数据再次提交。
>
> 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 请求一般会被缓存，比如常见的 CSS、JS、HTML 请求等都会被缓存；而 POST 请求默认是不进行缓存的。
>
> 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。

5、URL和URI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。

6、如何理解HTTP是无状态的?

> 当浏览器第一次发送数据给服务器时，服务器响应了；如果同个浏览器发起第二次请求给服务器时，他还是会响应；但是服务器不知道这就是刚才的那个浏览器。简而言之，服务器不会记住客户端是哪个，所以是无状态协议。

7、说说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，响应结束后就终端连接。HTTP1.0默认使用短连接。
>
> 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。

8、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。
>
> 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，从而导致多起建立TCP连接，大大消耗了资源。
>
> **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。

9、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（Keep-Alive: timeout=60）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**，则会发生侦测包，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。

10、HTTP1.1和HTTP2.0的区别?

> 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。

11、HTTP和HTTPS的区别?

> 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。
>
> HTTP和HTTPS用的端口不一样，HTTP端口是80，HTTPS是443。
>
> HTTPS协议**需要到CA机构申请证书**，一般需要一定的费用。
>
> 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。

12、什么是数字证书?

> 服务器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。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。

13、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。
> 2. 服务端响应Server Hello，告诉客户端服务端选中的加密算法。
> 3. 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。
> 4. 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。
> 5. 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。
> 6. 开始传输数据，使用同一个对称密钥来加解密。

14、DNS域名搜索的解析过程？ -----

> 1、浏览器搜索**自己的DNS缓存**
>
> 2、若没有，则搜索**操作系统中的DNS缓存和hosts文件**
>
> 3、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
>
> 4、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**
>
> 5、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
>
> 6、浏览器得到域名对应的IP地址

15、浏览器中输入URL返回页面的过程?

> 1、**解析域名**，找到主机 IP。
>
> 2、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。
>
> 3、建立 TCP 连接后，浏览器向主机发起一个HTTP请求。
>
> 4、参数从客户端传递到服务器端。
>
> 5、服务器端得到客户端参数之后，进行相应的业务处理，再将结果封装成 HTTP 包，返回给客户端。
>
> 6、服务器端和客户端的交互完成，断开 TCP 连接（4 次挥手）。
>
> 7、浏览器**解析响应内容，进行渲染**，呈现给用户。

16、DNS域名解析的过程?

> 在网络中定位是依靠 IP 进行身份定位的，所以 URL 访问的第一步便是先要得到服务器端的 IP 地址。而得到服务器的 IP 地址需要使用 DNS（Domain Name System，域名系统）域名解析，DNS 域名解析就是通过 URL 找到与之相对应的 IP 地址。
>
> DNS 域名解析的大致流程如下：
>
> 1. 先检**查浏览器中的 DNS 缓存**，如果浏览器中有对应的记录会直接使用，并完成解析；
> 2. 如果浏览器没有缓存，那就去**查询操作系统的缓存**，如果查询到记录就可以直接返回 IP 地址，完成解析；
> 3. 如果操作系统没有 DNS 缓存，就会去**查看本地 host 文件**，Windows 操作系统下，host 文件一般位于 "C:\Windows\System32\drivers\etc\hosts"，如果 host 文件有记录则直接使用；
> 4. 如果本地 host 文件没有相应的记录，会**请求本地 DNS 服务器**，本地 DNS 服务器一般是由本地网络服务商如移动、联通等提供。通常情况下可通过 DHCP 自动分配，当然也可以自己手动配置。目前用的比较多的是谷歌提供的公用 DNS 是 8.8.8.8 和国内的公用 DNS 是 114.114.114.114。
> 5. 如果本地 DNS 服务器没有相应的记录，就会**去根域名服务器查询**了。为了能更高效完成全球所有域名的解析请求，根域名服务器本身并不会直接去解析域名，而是会把不同的解析请求分配给下面的其他服务器去完成。

17、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。
>
> **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。

18、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。

19、什么是对称加密、什么是非对称加密?

> **对称加密**：通信双方使用**相同的密钥**进行加密解密。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`。

20、TCP Keepalive和HTTP-Alive是一个东西吗?

> - HTTP 的 Keep-Alive，是由**应用层（用户态）** 实现的，称为 HTTP 长连接。可以让多个http请求使用同一个tcp连接。这样做，可以减少tcp连接资源的开销。
> - TCP 的 Keepalive，是由 **TCP 层（内核态）** 实现的，称为 TCP 保活机制。如果对端程序是正常工作的。当 TCP 保活的探测报文发送给对端, 对端会正常响应，这样 **TCP 保活时间会被重置**，等待下一个 TCP 保活时间的到来。如果对端主机宕机（*注意不是进程崩溃，进程崩溃后操作系统在回收进程资源的时候，会发送 FIN 报文，而主机宕机则是无法感知的，所以需要 TCP 保活机制来探测对方是不是发生了主机宕机*），或对端由于其他原因导致报文不可达。当 TCP 保活的探测报文发送给对端后，石沉大海，没有响应，连续几次，达到保活探测次数后，**TCP 会报告该 TCP 连接已经死亡**。
>
> 注意，应用程序若想使用 TCP 保活机制需要通过 socket 接口设置`SO_KEEPALIVE`选项才能够生效，如果没有设置，那么就无法使用 TCP 保活机制。

21、如何在Linux中查看TCP状态?

> `netstat`命令用于查看网络连接、路由表、接口统计和多播成员等信息。
>
> netstat -a // 查看所有的网络连接，包括tcp和udp
>
> netstat -antp // 查看所有的TCP连接，并包括连接状态、本地地址和远程地址、端口号以及与每个连接关联的进程信息。
>
> netstat -r // 显示系统的路由表，包括目标网络、网关、接口和其他信息。
>
> netstat -i // 显示系统中所有网络接口的信息，包括接口名称、接收和发送的数据量等。
>
> netstat -s // 显示各种网络统计数据，如传输、错误、丢包等。
>
> netstat -t //`-t`参数将显示所有 TCP 连接。
>
> netstat -tuln | grep <端口号> // 显示指定端口上的监听情况

22、详解https

22.1 对称加密和非对称加密

> 对称加密：使用相同的密钥加解密，算法相对高效，适用于大量数据的加密场景，算法公开，安全性取决于密钥大小，但是密钥越大效率越低，需要权衡在安全和效率中做权衡。这个算法本身安全，但是使用场景不够安全，应该加密解密都使用同一个密钥。
>
> 非对称加密：使用匹配的一对密钥来分别进行加密和加密，这两个密钥是公开密钥和私有密钥。使用一方加密的数据只能通过另一方解密。它的特点是安全性高；但是加密解密复杂度高，效率低，并不适合频繁的数据交互。主要有两种用法：1、对数据加密；2、签名，证明数据是谁发的。
>
> 签名中的场景：假设James和Linda通信，并且Linda持有James的公钥。James要发送数据给Linda，James 用自己的私钥对明文的hash值进行加密，把密文(签名)和明文一起发给 Linda。Linda收到数据后，用James的公钥解密签名得到一个hash值，然后用收到的明文得到的hash值，如果两者一致就说明数据是James发的。

22.2 https协议

> https采用了非对称加密和对称加密的算法来解决数据传输的安全问题。非对称加密用来解决获取服务器公钥和发送会话密钥的安全性问题，对称加密则用来解决业务数据的交互安全性问题。
>
> 基本流程：
>
> 1、操作系统内置权威证书认证机构CA的证书X，证书中有认证机构的公钥数据。
>
> 2、网站A获取来自CA的认证证书。
> ** 服务器首先生成自己的公钥和私钥
> ** 找到认证机构生成网站A的证书，并保存在服务器中（证书中包含有网站的域名，证书有效期，证书颁发机构，网站自身的公钥等数据）
>
> 3、浏览器请求获取网站A证书
>
> 4、获取网站A证书数据后，用机构证书X解密网站A证书。（解密成功，获取网站A的密钥，说明是机构认证的）
>
> 5、浏览器将自己的密钥（对称密钥）发送给服务器（使用对称加密算法生成会话密钥B）(使用网站A的公钥对会话密钥B做加密处理，并发送给服务器)
>
> 6、服务器收到数据后，使用自己的私钥解密得到会话密钥
>
> 7、浏览器和服务器开始通信并使用会话密钥B加密解密。