字节跳动最爱考的前端面试题：计算机网络基础

> 注意：每道题前面出现的 (xx) 数字代表这道题出现的频次，此 计算机网络 基础是基于 30+ 篇前端面经整理出的问题和对应的回答、参考链接等。文章内容为拿到 Offer 的本人整理。

（3）问：HTTP 缓存

HTTP 缓存又分为强缓存和协商缓存：

- 首先通过 Cache-Control 验证强缓存是否可用，如果强缓存可用，那么直接读取缓存
- 如果不可以，那么进入协商缓存阶段，发起 HTTP 请求，服务器通过请求头中是否带上 If-Modified-Since 和 If-None-Match 这些条件请求字段检查资源是否更新：
  - 若资源更新，那么返回资源和 200 状态码
  - 如果资源未更新，那么告诉浏览器直接使用缓存获取资源

（5）问：HTTP 常用的状态码及使用场景?

- 1xx：表示目前是协议的中间状态，还需要后续请求
- 2xx：表示请求成功
- 3xx：表示重定向状态，需要重新请求
- 4xx：表示请求报文错误
- 5xx：服务器端错误

常用状态码：

- 101 切换请求协议，从 HTTP 切换到 WebSocket
- 200 请求成功，有响应体
- 301 永久重定向：会缓存
- 302 临时重定向：不会缓存
- 304 协商缓存命中
- 403 服务器禁止访问
- 404 资源未找到

- 400 请求错误
- 500 服务器端错误
- 503 服务器繁忙

你知道 302 状态码是什么嘛？你平时浏览网页的过程中遇到过哪些 302 的场景？

而 302 表示临时重定向，这个资源只是暂时不能被访问了，但是之后过一段时间还是可以继续访问，一般是访问某个网站的资源需要权限时，会需要用户去登录，跳转到登录页面之后登录之后，还可以继续访问。

301 类似，都会跳转到一个新的网站，但是 301 代表访问的地址的资源被永久移除了，以后都不应该访问这个地址，搜索引擎抓取的时候也会用新的地址替换这个老的。可以在返回的响应的 location 首部去获取到返回的地址。301 的场景如下：

- 比如从 baidu.com，跳转到 baidu.com
- 域名换了

（2）问：HTTP 常用的请求方式，区别和用途？

http/1.1 规定如下请求方法：

- GET：通用获取数据
- HEAD：获取资源的元信息
- POST：提交数据
- PUT：修改数据
- DELETE：删除数据
- CONNECT：建立连接隧道，用于代理服务器
- OPTIONS：列出可对资源实行的请求方法，常用于跨域
- TRACE：追踪请求-响应的传输路径

（）问：你对计算机网络的认识怎么样

> 应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

（3）问：HTTPS 是什么？具体流程

HTTPS 是在 HTTP 和 TCP 之间建立了一个安全层，HTTP 与 TCP 通信的时候，必须先进过一个安全层，对数据包进行加密，然后将加密后的数据包传送给 TCP，相应的 TCP 必须将数据包解密，才能传给上面的 HTTP。

浏览器传输一个 client_random 和加密方法列表，服务器收到后，传给浏览器一个 server_random、加密方法列表和数字证书（包含了公钥），然后浏览器对数字证书进行合法验证，如果验证通过，则生成一个 pre_random，然后用公钥加密传给服务器，服务器用 client_random、server_random 和 pre_random ，使用公钥加密生成 secret，然后之后的传输使用这个 secret 作为秘钥来进行数据的加解密。

（4）问：三次握手和四次挥手

为什么要进行三次握手：为了确认对方的发送和接收能力。

三次握手

三次握手主要流程：

- 一开始双方处于 CLOSED 状态，然后服务端开始监听某个端口进入 LISTEN 状态
- 然后客户端主动发起连接，发送 SYN，然后自己变为 SYN-SENT，seq = x
- 服务端收到之后，返回 SYN seq = y 和 ACK ack = x + 1（对于客户端发来的 SYN），自己变成 SYN-REVD
- 之后客户端再次发送 ACK seq = x + 1, ack = y + 1给服务端，自己变成 EASTABLISHED 状态，服务端收到 ACK，也进入 ESTABLISHED

> SYN 需要对端确认，所以 ACK 的序列化要加一，凡是需要对端确认的，一点要消耗 TCP 报文的序列化

为什么不是两次?

> 无法确认客户端的接收能力。

如果首先客户端发送了 SYN 报文，但是滞留在网络中，TCP 以为丢包了，然后重传，两次握手建立了连接。

等到客户端关闭连接了。但是之后这个包如果到达了服务端，那么服务端接收到了，然后发送相应的数据表，就建立了链接，但是此时客户端已经关闭连接了，所以带来了链接资源的浪费。

为什么不是四次?

四次以上都可以，只不过 三次就够了

四次挥手

- 一开始都处于 ESTABLISH 状态，然后客户端发送 FIN 报文，带上 seq = p，状态变为 FIN-WAIT-1
- 服务端收到之后，发送 ACK 确认，ack = p + 1，然后进入 CLOSE-WAIT 状态
- 客户端收到之后进入 FIN-WAIT-2 状态
- 过了一会等数据处理完，再次发送 FIN、ACK，seq = q，ack = p + 1，进入 LAST-ACK 阶段
- 客户端收到 FIN 之后，客户端收到之后进入 TIME_WAIT（等待 2MSL），然后发送 ACK 给服务端 ack = 1 + 1
- 服务端收到之后进入 CLOSED 状态

客户端这个时候还需要等待两次 MSL 之后，如果没有收到服务端的重发请求，就表明 ACK 成功到达，挥手结束，客户端变为 CLOSED 状态，否则进行 ACK 重发

为什么需要等待 2MSL（Maximum Segement Lifetime）：

因为如果不等待的话，如果服务端还有很多数据包要给客户端发，且此时客户端端口被新应用占据，那么就会接收到无用的数据包，造成数据包混乱，所以说最保险的方法就是等服务器发来的数据包都死翘翘了再启动新应用。

- 1个 MSL 保证四次挥手中主动关闭方最后的 ACK 报文能最终到达对端
- 1个 MSL 保证对端没有收到 ACK 那么进行重传的 FIN 报文能够到达

为什么是四次而不是三次?

** 如果是三次的话，那么服务端的 ACK 和 FIN 合成一个挥手，那么长时间的延迟可能让 TCP 一位 FIN 没有达到服务器端，然后让客户的不断的重发 FIN

参考资料

- zhuanlan.zhihu.com/p/86426969
- juejin.cn/post/684490…

问：在交互过程中如果数据传送完了，还不想断开连接怎么办，怎么维持?

在 HTTP 中响应体的 Connection 字段指定为 keep-alive

你对 TCP 滑动窗口有了解嘛?

在 TCP 链接中，对于发送端和接收端而言，TCP 需要把发送的数据放到**发送缓存区**, 将接收的数据放到**接收缓存区**。而经常会存在发送端发送过多，而接收端无法消化的情况，所以就需要流量控制，就是在通过接收缓存区的大小，控制发送端的发送。如果对方的接收缓存区满了，就不能再继续发送了。而这种流量控制的过程就需要在发送端维护一个发送窗口，在接收端维持一个接收窗口。

TCP 滑动窗口分为两种: **发送窗口**和**接收窗口**。

参考资料

- juejin.cn/post/684490…

问：WebSocket与Ajax的区别

本质不同

Ajax 即异步 JavaScript 和 XML，是一种创建交互式网页的应用的网页开发技术

websocket 是 HTML5 的一种新协议，实现了浏览器和服务器的实时通信

生命周期不同：

- websocket 是长连接，会话一直保持

- ajax 发送接收之后就会断开

适用范围：

- websocket 用于前后端实时交互数据
- ajax 非实时

发起人：

- AJAX 客户端发起
- WebSocket 服务器端和客户端相互推送

了解 WebSocket 嘛?

长轮询和短轮询，WebSocket 是长轮询。

具体比如在一个电商场景，商品的库存可能会变化，所以需要及时反映给用户，所以客户端会不停的发请求，然后服务器端会不停的去查变化，不管变不变，都返回，这个是短轮询。

而长轮询则表现为如果没有变，就不返回，而是等待变或者超时（一般是十几秒）才返回，如果没有返回，客户端也不需要一直发请求，所以减少了双方的压力。

参考链接

- www.jianshu.com/p/3fc3646fa...

HTTP 如何实现长连接？在什么时候会超时?

通过在头部（请求和响应头）设置 Connection: keep-alive，HTTP1.0协议支持，但是默认关闭，从HTTP1.1协议以后，连接默认都是长连接

- HTTP 一般会有 httpd 守护进程，里面可以设置 keep-alive timeout，当 tcp 链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 链接之后，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，就会丢弃该链接。
  - tcp_keepalive_intvl = 15

- tcp_keepalive_probes = 5
- tcp_keepalive_time = 1800

实际上 HTTP 没有长短链接，只有 TCP 有，TCP 长连接可以复用一个 TCP 链接来发起多次 HTTP 请求，这样可以减少资源消耗，比如一次请求 HTML，可能还需要请求后续的 JS/CSS/图片等

参考链接

- blog.csdn.net/weixin_3767...
- www.jianshu.com/p/3fc3646fa...

问：Fetch API与传统Request的区别

- fetch 符合关注点分离，使用 Promise，API 更加丰富，支持 Async/Await
- 语意简单，更加语意化
- 可以使用 isomorphic-fetch ，同构方便

参考资源

- github.com/camsong/blo...

（2）问：POST一般可以发送什么类型的文件，数据处理的问题

- 文本、图片、视频、音频等都可以
- text/image/audio/ 或 application/json 等

问：TCP 如何保证有效传输及拥塞控制原理。

- tcp 是面向连接的、可靠的、传输层通信协议

可靠体现在：有状态、可控制

- 有状态是指 TCP 会确认发送了哪些报文，接收方受到了哪些报文，哪些没有收到，保证数据包按序到达，不允许有差错
- 可控制的是指，如果出现丢包或者网络状况不佳，则会跳转自己的行为，减少发送的速度或者重发

所以上面能保证数据包的有效传输。

拥塞控制原理

原因是有可能整个网络环境特别差，容易丢包，那么发送端就应该注意了。

主要用三种方法：

- 慢启动阈值 + 拥塞避免
- 快速重传
- 快速回复

慢启动阈值 + 拥塞避免

对于拥塞控制来说，TCP 主要维护两个核心状态：

- 拥塞窗口（cwnd）
- 慢启动阈值（ssthresh）

> 在发送端使用拥塞窗口来控制发送窗口的大小。

然后采用一种比较保守的慢启动算法来慢慢适应这个网络，在开始传输的一段时间，发送端和接收端会首先通过三次握手建立连接，确定各自接收窗口大小，然后初始化双方的拥塞窗口，接着每经过一轮 RTT（收发时延），拥塞窗口大小翻倍，直到达到慢启动阈值。

然后开始进行拥塞避免，拥塞避免具体的做法就是之前每一轮 RTT，拥塞窗口翻倍，现在每一轮就加一个。

快速重传

在 TCP 传输过程中，如果发生了丢包，接收端就会发送之前重复 ACK，比如 第 5 个包丢了，6、7 达到，然后接收端会为 5，6，7 都发送第四个包的 ACK，这个时候发送端受到了 3 个重复的 ACK，意识到丢包了，就会马上进行重传，而不用等到 RTO （超时重传的时间）

选择性重传：报文首部可选性中加入 SACK 属性，通过 left edge 和 right edge 标志那些包到了，然后重传没到的包

快速恢复

如果发送端收到了 3 个重复的 ACK，发现了丢包，觉得现在的网络状况已经进入拥塞状态了，那么就会进入快速恢复阶段：

- 会将拥塞阈值降低为 拥塞窗口的一半
- 然后拥塞窗口大小变为拥塞阈值
- 接着 拥塞窗口再进行线性增加，以适应网络状况

问：OPTION是干啥的？举个用到OPTION的例子？

旨在发送一种探测请求，以确定针对某个目标地址的请求必须具有怎么样的约束，然后根据约束发送真正的请求。

比如针对跨域资源的预检，就是采用 HTTP 的 OPTIONS 方法先发送的。用来处理跨域请求

问：http知道嘛？哪一层的协议？（应用层）

- 灵活可扩展，除了规定空格分隔单词，换行分隔字段以外，其他都没有限制，不仅仅可以传输文本，还可以传输图片、视频等任意资源
- 可靠传输，基于 TCP/IP 所以继承了这一特性
- 请求-应答，有来有回
- 无状态，每次 HTTP 请求都是独立的，无关的、默认不需要保存上下文信息

缺点：

- 明文传输不安全
- 复用一个 TCP 链接，会发生对头拥塞
- 无状态在长连接场景中，需要保存大量上下文，以避免传输大量重复的信息

问：OSI七层模型和TCP/IP四层模型

- 应用层
- 表示层
- 会话层
- 传输层
- 网络层

- 数据链路层
- 物理层

TCP/IP 四层概念：

- 应用层：应用层、表示层、会话层：HTTP
- 传输层：传输层：TCP/UDP
- 网络层：网络层：IP
- 数据链路层：数据链路层、物理层

（3）问：TCP 协议怎么保证可靠的，UDP 为什么不可靠?

- TCP 是面向连接的、可靠的、传输层通信协议
- UDP 是无连接的传输层通信协议，继承 IP 特性,基于数据报

为什么 TCP 可靠？TCP 的可靠性体现在有状态和控制

- 会精准记录那些数据发送了，那些数据被对方接收了，那些没有被接收，而且保证数据包按序到达，不允许半点差错，这就是有状态
- 当意识到丢包了或者网络环境不佳，TCP 会根据具体情况调整自己的行为，控制自己的发送速度或者重发，这是可控制的

反之 UDP 就是无状态的和不可控制的

HTTP 2 改进

改进性能：

- 头部压缩
- 多路信道复用
- Server Push