计网八股

HTTP介绍

- http是超文本传输协议，总结来说就是在计算机世界实现两点之间传输文本、音频、图片等超文本内容的约定和规范

http缓存机制

- http缓存是一种可以临时存储网页资源，以减少服务器延迟的技术
- 强缓存
  - 客户端直接使用缓存中的资源，不从服务器拉取新资源，也不验证当前的缓存资源是否过期。返回状态码为200
  - http1.0使用Expires进行强缓存，它的值为服务器返回给客户端的一个资源过期的GMT时间（格林尼治标准时间），当下一次客户端请求时会判断当前客户端的GMT时间是否小于expires，若小于，则直接使用缓存，否则向服务器请求新的资源。但是如果客户端更改过自己的GMT时间，则会出现缓存命中的误差。并且GMT时间是由格林尼治天文台每隔1小时发放，也存在误差。
  - http1.1中使用了Cache-Control进行强缓存，它可以在**请求头和响应头**中使用，并且配置了丰富的属性
    - no-store，表示不使用任何缓存，**具有http缓存的最高优先级**
    - no-cache,表示不使用强缓存
    - public,共有缓存，任何从源服务器到客户端的节点都可以对资源缓存
    - private,私有缓存， 只有当前客户端可以对资源进行缓存
    - max-age,客户端可以缓存资源的最长时间，单位为秒。**优先级高于Expires**。如果客户端缓存资源的时间小于该值，则使用缓存，否则进入Expires流程
    - s-max-age,代理缓存服务器最长的缓存时间，单位秒。优先级高于max-age和Expires，仅适用于缓存服务器
- 协商缓存
  - 客户端会向服务器验证资源有效性，服务端收到请求后，会先看**请求头中是否有no-cache**,如果有，则返回新的资源；如果有效，则返回304，客户端可以使用缓存，否则则返回新的资源。相对于无缓存流程的优势在于仅响应状态码后，客户端直接从本地缓存获取文件，而**无需进行文件下载**。减少了网络响应的文件大小，进而加快了网络响应速度。
  - http1.0中，客户端第一次向服务器请求资源时，服务器会返回资源。同时会在响应头中添加Last-Modified字段来表明资源的最后修改时间。当客户端强制缓存失效后，会重新向服务器进行缓存有效性验证。在验证的请求头中，会添加If-Modified-Since字段。服务器会对请求头中的If-Modified-Since和其存储的资源Last-Modified进行比较。若If-Modified-Since的时间不小于Last-Modified，则资源有效，返回304（Not Modified）。否则返回资源本身，并且重新记录文件的Last-Modified。
    - 这种方式只会关心资源文件的最后修改时间，而不关心资源内容。如果**资源文件修改后又恢复了，则客户端无法使用缓存。**
  - http1.1的使用的Etag字段**优先级高于last-Modified**,是响应头的资源标识符，值由服务器生成，通常是使用**内容的散列或简单的使用版本号。**服务器接收到浏览器请求后，会先进行If-None-Match与ETag值的比较。若相等，则资源有效，返回304（Not Modified）。否则返回资源本身，并且重新记录文件的ETag。

    - If-None-Match：请求头携带的是否无匹配文件字段。**优先级高于Last-Modified**。当服务器没有任何资源的ETag与请求头携带的ETag值完全一样时，返回最新的资源，否则服务器会返回304
    - If-Match：请求头携带的是否存在匹配文件字段。对于简单请求需要搭配 Range首部使用。对于非简单请求，如PUT，可用于上传ETag

http请求头都有哪些呢

- Accept
    - 指定能够接受的响应内类型
- Accept-Charset
    - 指定能够接受的字符串类型
- Accept-Encoding
    - 用于指定能够接受的编码方式列表
- Accept-language
    - 用于指定能够接受的回应内容的自然语言列表
- Authorization
    - 用于超文本传输协议的认证的认证信息，通常用于进行用户身份验证
- Cache-Control
    - 用来指定在请求/响应链中的所有缓存机制都必须遵守的指令，例如no-cache表示不缓存
- Cookie
    - 标识浏览器保存的cookie信息
- Content-length
    - 表示请求体的长度，以八位字节数组（8位的字节）表示
- Content-Type
    - 用于指定请求体的多媒体类型，例如application/x-www-form-urlencoded表示表单数据
- User-Agent
    - 表示浏览器的浏览器身份标识字符串，用于标识用户使用的浏览器信息
- Host
    - 表示服务器的域名，以及服务器所监听的传输控制协议端口号
- Referer
    - 表示请求的来源页面，即用户是从哪个页面跳转过来的
- Origin
    - 用于发起一个针对跨来源资源共享的请求，表示允许跨域请求
- Connection
    - 客户端要求服务器使用的连接方式，通常是持久连接/长连接即Keep-Active(http1.1自带)
- If-Modified-Since
    - 标识当前请求资源最近的一次更新

HTTP响应头有哪些

- Date
  - 表示当前资源发送的服务器日期与时间
- Last-Modified
  - 表示当前响应资源最后被修改的服务器时间
- Transfer-Encoding
  - 表示当前响应资源传输实体的编码格式
- Set-Cookie
  - 设置Cookie信息
- Location
  - 在重定向中或创建新资源时使用
- Server
  - 表示服务器名称
- Content-length
  - 响应资源的字段长度
- Content-Type
  - 响应资源的数据类型

http请求和响应报文格式

- 请求/报文由四部分组成：请求行、请求头、空行、报文主体数据
- 请求报文
  - 请求报文的请求行包括：请求方法、请求地址（url）、协议版本
    - 请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT
- 请求头按照"键: 值"的格式（冒号后由一个空格）
- 请求体是 POST 请求方式中的请求参数，以 `key = value` 形式进行存储，多个请求参数之间用 `&` 连接，如果请求当中请求体，那么在请求头当中的 Content-Length 属性记录的就是该请求体的长度。
  - **任意请求体**：移动开发者常见的，请求体是任意类型的，服务器不会解析请求体，请求体的处理需要自己解析，如 POST、JSON 的时候就是这类
  - **查询字符串**：URL 中 Query String 的格式要求，多个键值对之间用 `&` 连接，键与值之间用 `=` 连接，且只能用 ASCII 字符，非 ASCII 字符需使用 `UrlEncode` 编码
  - **文件上传**：当需要实现 **文件上传** 时，请求体会被分成多个部分，每个字段 / 文件都被首部字段 `Content-Type` 的 **boundary** 指令指定的值分成单独的段，每段以 `--` 加 `boundary` 指令的值开头，然后是该段的描述头，描述头之后空一行接内容，请求结束的标识为 `boundary` 后面加 `--`。区分是否被当成文件的关键是 `Content-Disposition` 是否包含 `filename`，因为文件有不同的类型，所以还要使用 `Content-Type` 指示文件的类型，如果不知道是什么类型取值可以为 `application/octet-stream` 表示文件是一个二进制的文件，如果不是文件则 `Content-Type` 可以省略。
- 响应报文
  - 响应报文的响应行包括：http版本、状态码、短语
  - 响应头按照"键: 值"的格式（冒号后由一个空格）
  - 服务器返回的响应资源数据

Content-Type有哪些

- text/html
- text/plain
  - 纯文本格式
- text/css
- text/javascript
- image/gif
- image/jpeg
- image/png
- application/x-www-form-urlencoded
  - POST专用：普通的表单默认提交是这种方式。form表单数据被编码为key/value格式发送到服务器
- application/json
  - POST专用：用来告诉服务端消息主体是序列化后的JSON字符串
- text/xml
  - POST专用，发送xml数据
- multipart/form-data
  - POST专用，用以支持向服务器发送二进制数据，以便可以在POST请求中实现文件上传等功能

http状态码

- 1xx --提示信息，是协议处理的中间状态
- 2xx --表示成功的处理了客户端的请求
  - 200 OK ,一个成功状态码，如果是非HEAD请求，服务器返回的响应头中会有body数据
  - 204 No Content ,也是成功状态码，但是响应头中没有body
  - 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出现了问题，需要重新指定URL，也就是重定向
  - 301 Moved Permanently ，永久重定向，表示请求的资源已经不在了，客户端需要用其他URL请求
  - 302 Found ，临时重定向，表示请求的资源还在但是暂时需要另外一个URL访问
  - 301和302都会在响应头里使用Location字段，里面写了新的URL，浏览器会自动重定向新的url
  - 304 Not Modified ,缓存重定向，不具备跳转的含义，表示资源未修改，重定向已存在的缓存资源，用户缓存控制
- 4xx --表示客户端发送的报文有误，服务器无法处理
  - 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常，服务器内部处理时出现了问题
  - 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，暂时无法响应服务器，类似“网络服务正忙，请稍后

http常见的字段有哪些

- HOST --客户端访问的指定服务器域名
- Content-Length --响应时表明本次响应的字段长度
- Connection --客户端要求服务器使用的连接方式，通常是持久连接/长连接即Keep-Active(http1.1自带)
- Content-Type --服务器响应告诉客户端本次返回的数据类型是什么
- Accept --客户端告诉服务器接受的数据类型是什么
- Content-Encoding --表示服务器返回的数据的压缩方式
- Accept-Encoding --客户端告诉服务器接受的压缩方式

http里的keep-alive在哪，用处是啥

- 用在http请求报文的请求体中的connection字段，用以表示客户端与服务端要建立持久连接/长连接

GET和POST区别

- GET请求是向服务器端请求获取数据，用于查询操作；POST是用来传输实体对象的，可以用来进行添加、修改、删除等操作。
- GET操作是幂等的，而POST则不是
- 在参数传递上，GET是将参数拼接到URL中的，而POST的参数是写在请求体中的
- GET请求一般是可以被缓存的，POST请求默认是不缓存的
- GET 请求的参数是通过 URL 传递的，而 URL 的长度是有限制的，通常为 2k，当然浏览器厂商不同、版本不同这个限制的大小值可能也不同，但相同的是它们都会对 URL 的大小进行限制；而 POST 请求参数是存放在请求正文（request body）中的，所以没有大小限制
- 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 请求的参数会保存在历史记录中，而 POST 请求的参数不会保留到历史记录中
- GET 请求的地址可被收藏为书签，而 POST 请求的地址不能被收藏为书签

http（1.1）优缺点

- 简单：http基本报文的格式是header+body，头部信息也是key-value的形式，易于理解
- 灵活与易于扩展：http协议里的请求方法、url、状态码、头字段等信息都支持程序员自定义。http作用于应用层（OSI第七层），它的下层可以随意改变
- 应用广泛和跨平台：http协议发展至今应用广泛，从台式机浏览器到移动端的各种app中都有使用，同时也有跨平台的特性
- 无状态
  - 服务器不会记忆http的状态，这样有利于减少服务器负担
  - 当客户在进行一系列关联性操作时，因为服务器没有保存http状态，所以每次都需要再验证一遍用户身份，例如：登录-加入购物车-下单-结算-支付
- 明文传输
- 不安全
  - 使用明文传输，传输信息易被泄露
  - 无法验证报文完整性，所以信息易被篡改

    - 不验证通信双方身份，可能会遭遇伪装

http和https区别

- http通信采用明文传输，信息易泄露，而https在TCP和HTTP应用层之间加了SSL/TLS安全协议，实现加密传输
- http连接建立相对连接，只需要TCP三次握手即可，https在经历TCP三次握手后还需要进行SSL/TLS握手，之后才可建立连接
- http端口号为80，https端口号为443
- https协议需要向CA（证书权威机构）申请数字证书，来确保服务器的身份可信

HTTPS解决了HTTP的哪些问题，以及是如何解决的

- HTTP是明文传输，存在信息易被窃听、信息易被篡改无法保证信息完整性、无法验证通信双方身份易被冒充。而HTTPS在HTTP应用层和TCP之间加入了SSL/TLS协议来解决上面问题
- 混合加密--HTTPS采用非对称加密和对称加密结合的【混合加密】的方式，在通信之间通过非对称加密的方式交换会话密钥，后续不再使用非对称加密；在通信过程中全部采用对称加密的会话密钥的方式加密明文数据。
- 摘要算法--在发送数据前通过摘要算法算出明文的独一无二的【指纹】，发的时候将明文+【指纹】一起加密为密文，发送给服务器，服务器解密后，用相同的摘要算法算出发送过来的明文，通过比较客户端携带的【指纹】和当前算出的【指纹】做比较，若【指纹】相同，说明数据是完整的。
- 数字证书--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。

  借助第三方权威机构 CA （数字证书认证机构），将服务器公钥放在数字证书（由数字证书认证机构颁发）中，只要证书是可信的，公钥就是可信的。

TLS/SSL1.2握手过程（RSA加密）

- 首先，客户端向服务端发送加密通信请求，向服务端发送内容：
    - 客户端支持的TLS/SSL协议
    - 客户端生成的一个用户后面生成秘钥的随机数
    - 客户端支持的密码套件列表
- 服务器收到客户端请求后，向客户端发出响应，相应内容包括：
    - 确认TLS/SSL协议，如果不支持则终止加密通信
    - 服务器生成的一个后面用于生成秘钥的随机数
    - 确认的密码套件列表
    - 数字证书
- 客户端在收到服务器响应后，首先从浏览器或操作系统的CA公钥，检测服务器的数字证书的真实性，如果证书没问题则从中提取出服务器的公钥，使用它加密报文，向服务器发送内容：
    - 一个被服务器公钥加密的随机数
    - 加密算法改变通知，表示之后的信息都会用会话秘钥加密通信
    - 客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时会把之前所有内容的发生数据做个摘要，用于服务器检测
    - 上面第一项的随机数是整个握手阶段的第三个随机数，这样客户端和服务器就会分别拥有三个随机数，接着就用双方协商的加密算法，各自生成本次通信的会话密钥
- 服务器在收到客户端的第三个随机数后，通过协商的加密算法，计算出本次通信的会话密钥。之后向客户端最后发送信息：

    - 加密算法改变通知，表示随后的信息都会用会话密钥加密通信
    - 服务器握手结束通知，表示服务器的握手阶段结束。这一项同时把之前所有的内容发生的数据做个摘要，用户客户端校验
- 至此，TLS握手阶段全部结束

http1.1性能怎么样

- http1.1较http1.0采用了持久连接/长连接的方式，使得客户端每次发送请求时不用像http1.0一样先建立TCP握手连接，只要任意一端没有提出断开连接，就保持TCP连接状态，减少了TCP连接断开的消耗，减轻了服务器负担
- 因为http1.1采用了长连接的方式，使得管道网络运输成为了可能，在同一个TCP连接中，客户端可以发送多个请求，且每个请求无需等待上一个请求响应后即可发送，减少了整体的响应时间
- 队头阻塞--http1.1采用的是【请求-应答】模式，当顺序发送的请求序列中的一个请求因某种原因被阻塞后，后面的请求也被阻塞，当值客户端出现了请求不到数据的现场。
- http1.1总体性能一般

http2.0特点（较1.1）

- （兼容http1.1
    - url里没有引入新的协议名，仍然是明文传输为http://，加密传输为https://
    - 只在应用层进行了改变，还是基于TCP协议传输。将HTTP分别为【语义】、【语法】两部分，【语法】层仍然和http1.1一样，【语法】层进行了大部分改变）
- 头部压缩
    - http2开发采用了HPACK算法，来对头部进行压缩处理，在客户端和服务器都建立一个的“字典”，该字典“静态表”部分包含了61位高频头部字符段，用索引号表示对应的重复的字符串，还采用了哈夫曼算法来对整数和字符串进行压缩，提高50%-90%的压缩效率
    - 如果有“静态表”中没有的内容，则在第一次发送后在“静态表”末尾创建“动态表”，请该字符段存入“动态表”中，进行更新。但是“动态表”如果过大占用的内存也会很大，所以需要服务器限制HTTP2的连接时间/请求次数
- 二进制帧
    - 将原来的【header+body】形式的报文划分成了headers frame和data frame两个帧，并且采用了二进制编码
    - 每个帧有帧头部和帧数据两个部分，其中帧头部的最后4个字节为流标识符（Stream ID）用来标识该 Frame 属于哪个 Stream，接收方可以根据这个信息从乱序的帧里找到相同 Stream ID 的帧，从而有序组装信息。帧数据存放的是经过HPACK压缩后的http头部和包体。
- 并发传输/多路复用
    - 同域名下所有通信都在单个连接上完成。
    - 单个连接可以承载任意数量的双向数据流(stream)。
    - 数据流以消息的形式发送，而消息又由一个或多个帧组成，多个帧之间可以乱序发送，因为根据帧首部的流标识可以重新组装。
    - 在HTTP/2中，每个请求都可以带一个31bit的优先值，0表示最高优先级， 数值越大优先级越低。有了这个优先值，客户端和服务器就可以在处理不同的流时采取不同的策略，以最优的方式发送流、消息和帧。
    - 如此可以解决http1.1中出现了队头堵塞问题。
- 服务器主动推送资源
    - 服务器支持主动推送资源，大大提升了消息的传输性能，服务器推送资源时，会先发送 PUSH_PROMISE

帧，告诉客户端接下来在哪个 Stream 发送资源，然后用偶数号 Stream 发送资源给客户端。

http2有什么缺点

- 队头堵塞
  - http2的多个请求是跑在一个TCP连接中的，TCP是字节流协议，必须保证收到的字节流数据是完整且有序的，如果一个序列号较低的请求丢失了，后面的序列号较高的请求即使已经被接收了，应用程序也无法从内核中读取到该请求的数据，
- TCP与TLS(SSL)的握手延迟
  - HTTP2是基于TCP协议来传输的，所以需要先经历TCP三次握手，消耗1.5RTT。如果还要使用https的话，还需要在TCP三次握手后经历TLS/SSL（1.2）四次握手建立连接，消耗2RTT，并且还会经历TCP慢启动的减速过程。这说明我们在传输数据之前就需要至少等待3-4RTT的时间
- 网络迁移需要重新连接
  - http2是基于TCP协议传输的，而一个TCP的连接是由四元组来确认的，一旦IP地址或端口改变，例如移动设备的用户切换网络，就需要重新进行TCP三次握手，TLS四次握手建立连接。
- 多路复用导致服务器压力上升
  - 多路复用没有限制同时请求数。请求的平均数量与往常相同，但实际会有许多请求短暂爆发，导致顺时QPS爆涨
- 多路复用容易Timeout
  - 大量请求同时发送，虽然http2连接内存在多个并行的数据流，但是网络带宽与服务器资源有限，可能会导致每个数据流内的请求被稀释，虽然它们开始都相差很短，但仍有可能会超时

HTTP3的特点（较http2）

- HTTP3基于UDP协议来传输，同时基于UDP协议在应用层实现了QUIC协议，使得基于UDP协议的不可靠运输也变得可靠起来
- 解决队头堵塞
  - UDP协议不会关注数据包的顺序与丢失问题，但QUIC为保证数据包的可靠性，每个数据包都会有一个**序号唯一标识**，当一个流中的某个包丢失，那么其他数据包到达了也无法被接收，直到QUIC重传丢失的报文，数据包才会被完整接收。而其他流的数据报文只要被完整接收，HTTP/3 就可以读取到数据。**QUIC 连接上的多个流都是独立互不影响的。**
- 更快建立连接
  - QUIC协议内集成**TLS1.3**，它在自己的帧会携带 TLS 里的"记录"，因此仅需 1 个 RTT 就可以「同时」完成建立连接与密钥协商，甚至在第二次连接的时候，应用数据包可以和 QUIC 握手信息（连接信息 + TLS 信息）一起发送，达到 0-RTT 的效果。
- 实现连接迁移
  - 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，没有卡顿，实现**连接迁移**

UDP和TCP的区别，分别对应哪些协议

- 建立连接的差异

    - TCP 是面向连接的，它的数据传输前需要经历三次握手建立一条虚拟连接，数据传输需要在这条虚拟连接上进行，数据传输完毕后需要断开这条连接；而 UDP 传输不是面向连接的，UDP 发送数据不会建立连接，也不会关心接收端的状态
- 可靠性和有序性的差异
    - TCP通过序列号、确认应答、重传机制、流量控制、拥塞控制等技术来保证发送的数据可以完整且有序的到达接收方；UDP不保证数据报的可靠传输，它发送的数据包可能会丢失、重复或到达顺序错乱，不提供错误恢复功能
- 报文段的差异
    - UDP报文段的固定头部只有8个字节，分别是源端口号、目的端口号、长度、校验和
        - 
    - TCP报文段的固定头部由20个字节，包括源端口号、目的端口号、序列号、确认序列号、首部长度、标志字段、接收窗口、校验和、紧急数据指针。除了固定首部外还有可选的选项字段
        - 
    - 总体TCP报文段的开销比UDP报文段大
- 传输效率的差异
    - **TCP**：由于需要进行连接管理、错误检测和恢复，以及维持连接状态，其数据传输速度相对较慢，协议开销较大。
    - **UDP**：由于不需要建立连接，且几乎没有错误恢复机制，使得UDP的数据传输速度较快，协议开销小，效率较高。
- 使用场景差异
    - TCP:适用于需要高可靠性的应用，如网页浏览、电子邮件、文件传输等
    - UDP:适用于对传输速度和效率要求高、可以容忍一定数据丢失的应用，如在线视频会议、实时游戏、流媒体等。
- http1.0/http1.1/http2.0都是基于TCP传输协议，而http3.0是基于UDP协议的

对称加密和非对称加密分别什么时候用到

- 对称加密，也称为单密钥加密，是指加密和解密使用同一个密钥。这种加密方式效率较高，适合于加密大量数据。
- 非对称加密，也称为公钥加密，是指加密和解密使用不同的密钥，即公钥和私钥。公钥用于加密数据，私钥用于解密数据。非对称加密的安全性较高，但效率较低，适合于加密少量数据。

TCP三次握手

- 客户端与服务器开始都处于close状态，服务器会先监听某个端口，进入Listen状态
- 这里让客户端为请求方，客户端会先生成一个序列号设为client_isn，将标志SYN设为1，生成并发送SYN报文，之后进入SYN_SENT状态
- 服务器收到客户端的SYN报文后，将ACK和SYN标志设为1，也会生成一个序列号设为server_isn，将确认序列号设为client_isn + 1,生成并发送ACK、SYN报文，之后进入SYN_RCVD状态
- 客户端收到服务器的ACK、SYN报文后会再向服务器发送一个ACK确认应答报文，之后进入ESTABLISHED状态
- 服务器收到客户端的ACK确认应答报文后进入ESTABLISHED状态。
- 至此TCP三次握手建立连接

TCP四次挥手

- 这里设客户端主动要求断开连接，其会先向服务器发送FIN报文，之后进入FIN_WAIT1状态
- 服务器收到客户端发来的FIN报文后向客户端发送ACK报文确认应答，之后进入CLOSE_WAIT状态
- 在服务器处理完当前数据后，会向客户端也发送一个FIN报文，之后进入LAST_ACK状态
- 客户端在收到服务器的ACK报文后，进入FIN_WAIT2状态
- 客户端在收到服务器的FIN报文后，会向服务器发送一个ACK确认应答报文，之后进入TIME_WAIT状态，经过2MSL时间后自动断开连接
- 服务器在收到客户端的ACK报文后进入close状态，断开连接

TCP三次握手是否可以改成两次

- 不可以
- 三次握手连接才可以阻止重复历史连接的初始化
    - 当客户端连续发送多个SYN报文时，旧的SYN报文比新的SYN先到达服务器端，服务器返回的ACK、SYN报文，客户端收到ACK、SYN报文后根据上下文判断出这是一个历史连接，就会发送RST报文给服务器来中断这次连接。如果只有两次握手，则无法判断当前连接是否为历史连接
- 三次握手连接才可以同步双方初始序列号
    - 客户端发送带有初始序列号的SYN报文后，需要服务器回复一个ACK报文来确认发送的SYN报文已被接收；同理，服务器发送的带有初始序列号的SYN报文，也需要客户端回复一个ACK报文来确认发送的SYN报文已被收到。这样双方初始序列号才能被可靠同步。如果是两次握手则只能保证一方的初始序列号被成功接收

当输入一个URL/网址到网页中显示，期间经历了什么

- 先对url进行解析，确定要请求的web服务器/域名和资源
- 根据确定得要请求得web服务器和资源后，生成http请求报文信息
- DNS解析
    - 先根据域名在本地的缓存（浏览器缓存->操作系统缓存->本地DNS服务器）中查找对应的IP地址
    - 如果在本地缓存中没有找到该域名对应的ip地址，则会向根域名服务器进行查询，根域名服务器不会解析域名，而是告诉我们找对应的顶级域名服务器；接着向顶级域名服务器进行查询，其会给出我们生成该域名的对应的权威域名服务器；接着我们向对应的权威域名服务器进行查询，获得对应的IP地址
    - 获得该域名得IP地址后，会将结果返回给本地DNS服务器，DNS服务器将IP地址返回给当前客户端
- TCP三次握手建立起客户端与服务端得连接，如果使用得是https协议，还需进行SSL/TSL协议握手后才会建立连接
- 建立起客户端和服务端得连接后，浏览器会向服务器发送http请求报文来获取所需得资源。服务器收到后，如果没有问题会返回http响应报文，该响应报文会返回客户端所请求得数据
- 当浏览器收到了服务器返回的HTML文件后，会对其进行下载解析
    - 首先对html元素进行解析，生成DOM树
    - 在解析文件中遇到css文件，则会下载css文件，下载完成后，解析css文件生成CSSOM，这一步不会阻塞对html的解析/DOM树的生成
    - 当遇到script脚本文件后，则会先下载解析当前的script脚本，会阻塞其后续html文档的解析。
    - 结合生成的DOM树和CSSOM树构建生成Render Tree(渲染树)
    - 在渲染树(Render Tree)上运行布局(Layout)以计算每个节点的几何体。
        - 渲染树会表示 要显示哪些节点以及其他样式，但是不表示每个节点的尺寸、位置等信息
        - 布局的主要目的是为了确定呈现树中所有节点的宽度、高度和位置信息

    - 将每个节点绘制(Paint)到屏幕上
        - 在绘制阶段，浏览器将布局阶段计算的每个frame转为屏幕上实际的像素点
        - 包括将元素的可见部分进行绘制，比如文本、颜色、边框、阴影、替换元素（比如img）
- 渲染完成后，会继续加载页面中的其他部分，如异步加载的内容/script脚本中的动态内容。如果没有其他资源需要加载了，则通过TCP四次挥手断开连接

WS（Websocket）和HTTP有什么区别

- **相同点**：都是一样基于TCP的，都是可靠性传输协议。都是应用层协议
- **联系**：WebSocket在建立握手时，数据是通过HTTP传输的。但是建立之后，在真正传输时候是不需要HTTP协议的。
- WebSocket是双向通信协议，模拟Socket协议，可以双向发送或接受信息，而HTTP是单向的；虽然HTTP/2也具备服务器推送功能，但HTTP/2 只能推送静态资源，无法推送指定的信息。
- WebSocket是需要浏览器和服务器握手进行建立连接的，而http是浏览器发起向服务器的连接。

跨域以及解决方法

-

路由分类

Cookie、Session、Token、JWT参考如下链接

-

iframe通信方法

流式传输，HTTP返回的是流式传输吗

请求边界条件

了解HTTP吗，具体说一说AES是对称加密还是非对称加密

HTTP缓存是如何实现的，Etag是怎样生成的

CDN的原理

XXS

头部压缩用什么算法

解释一下HPACK算法

多路复用解决了什么问题?

说一下强缓存和协商缓存

http3协议传输层里用了什么协议？udp有什么优点或者缺点?

常见的Web攻击以及防御手段有哪些

如何用离线包来解决首屏渲染时间