(2) 前端基础 (1) HTTP/HTML/浏览器1

一 HTTP/HTML/浏览器
1 说一下 http 和 https【注意加上 http1.0、1.1、2.0 后的对比呢？？？】
1.1 http，超文本传输协议，是互联网上应用最为广泛的一种网络协议。
是一个 客户端 和 服务端 请求和应答的标准（TCP）。可以是浏览器更加高效，网络传输更少！！
https【安全版的http，在“表示层加入了ssl（secure sockets layers 安全套接层）；TLS 传输安全层 transport layer security 是其继任者），它们是为网络通信提供 安全 及 数据完整性 的一种安全协议。
TLS与SSL（“它们应该位于表示层，加密表示呗～”） 是在 传输层 与 应用层 之间对网络连接进行加密“】，简单来讲就是安全版的 http。

https主要作用是： 建立一个信息的安全通道、确保数据的传输和确保网站的真实性！！

1.2 主要区别：
https需要 ca 证书【全称是 certificate authority 即证书授权中心。“看作一种介绍信”。】，费用较高。
使用不同的链接方式，端口也不同。一本来说，http是80端口，https是443端口。
补充：CA数字证书在用户公钥后附加了用户信息及CA的签名。公钥是密钥对的一部分，另一部分是私钥。公钥公之于众，谁都可以使用。私钥只有自己知道。由公钥加密的信息只能由与之相对应的私钥解密。为确保只有某个人才能阅读自己的信件，发送者要用收件人的公钥加密信件；收件人便可用自己的私钥解密信件。同样，为证实发件人的身份，发送者要用自己的私钥对信件进行签名；收件人可使用发送者的公钥对签名进行验证，以确认发送者的身份。

http连接简单、无状态；https是由 HTTP+SSL 协议构建的加密传输、身份认证的网络协议，比如 http协议安全【加密、数据完整性、身份验证等有所保证了，”http对应的 3缺点“】。

https是超文本传输协议，信息不经加密、直接明文传输【容易被窃取、篡改等！！】；
https则是具有安全性的 SSL加密传输协议。

1.3 https协议的工作原理【较复杂，截图。。。】：

客户使用https url访问服务器，则要求web 服务器建立ssl链接。

web服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），返回或者说传输给客户端。

客户端和web服务器端开始协商SSL链接的安全等级，也就是加密等级。

客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。

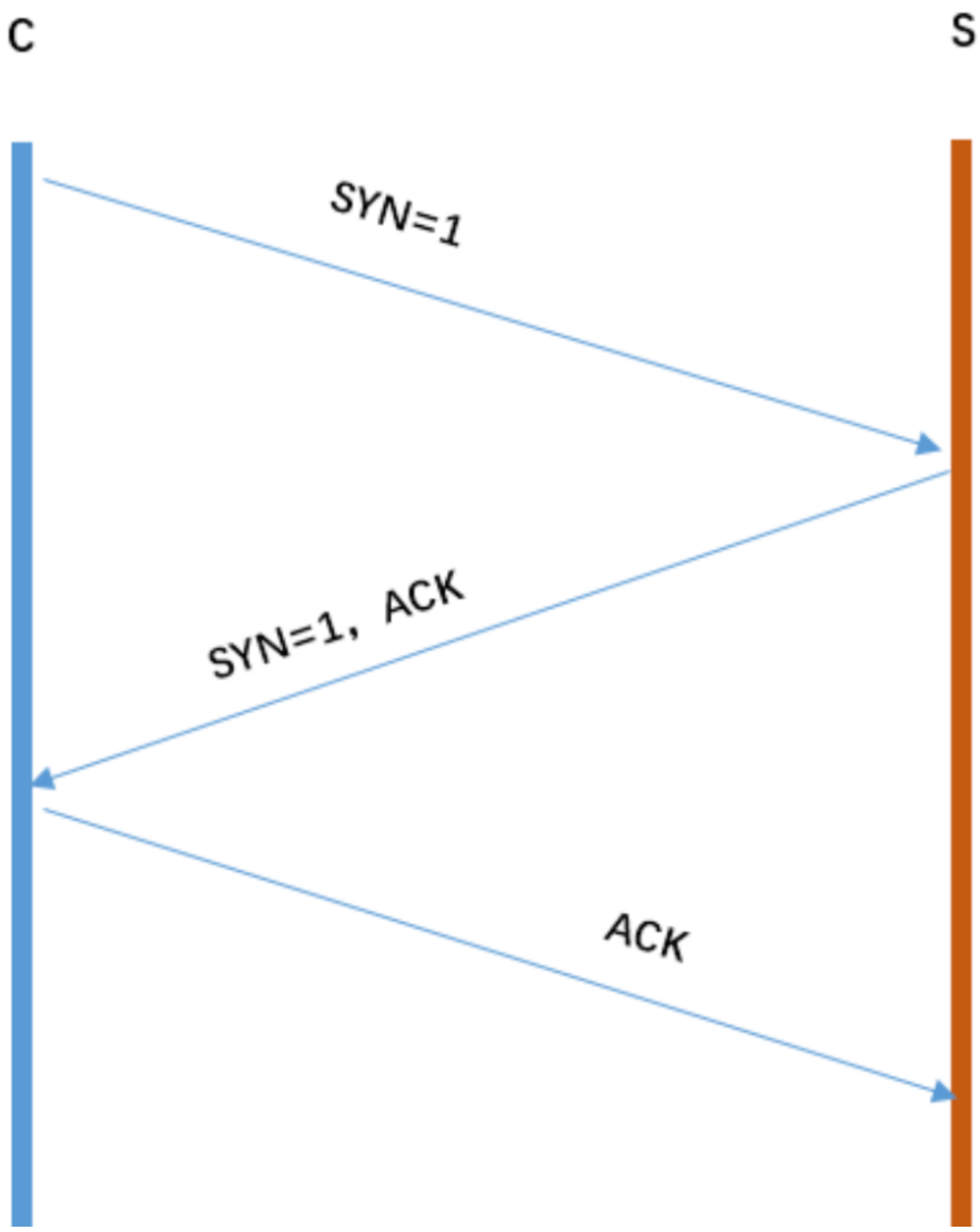
web服务器通过自己的私钥解密出会话密钥。

web服务器通过会话密钥加密与客户端之间的通信。

1.4 https优点：
使用https可以 认证 用户和服务器， 确保数据正确发送到合适的地址！！ 。
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，可确保数据的 安全性、完整性等。
https是当下最安全的解决方案，但不是绝对安全！！ 它增加了中间人攻击的成本。
谷歌搜索引擎算法，采用https排名相对会靠前一些！！！

1.5 https缺点
握手耗时，会使页面加载时间延长50%,增加10-20%的耗电。
https缓存【还有 https缓存一说？？？】没有 http高效，会增加数据开销。
SSL证书【如ca证书】也需要钱。
SSL证书【一个SSL证书、一个IP 严格对应？？】需要绑定一个ip。不能同一个ip上绑定多个域名，ipv4资源支持不了这种消耗【？？？】。

2 简要概括说明 三次握手【确认 C、S双方 都可以正常的发送、接收报文】
第一次握手：说明 服务端S 可以接受 C发送过来的报文；
第二次握手：说明 S收到了 C发送的报文；并且确定 自己可以接收到 S发送过来的报文。
第三次握手：S 确认 C收到了自己发送的报文。
总之，各阶段的握手功能是：
1 S知道C的发送功能正常。
2 C知道 S的接收 和 发送 功能正常。
3 S知道C的接收功能正常。



3 TCP(传输控制协议) 和 UDP(用户数据包协议)

3.1 TCP是 传输控制协议，UDP是 用户数据报协议。

3.2 TCP是面向连接的，提供可靠的【其传输的数据是无差错、不丢失、不重复，且按序到达。编号好了可以不用按序抵达呀、到时接收方自行组装即可？？？！】服务、适合 大数据量的交换！！UDP不可靠。

3.3 TCP是面向连接的，UDP是无连接的、即发送数据之前无需建立连接。

3.4 TCP面向字节流【定义？？ 流？？】；UDP面向报文、网路出现拥塞也不会使得速率下

降【对应实时的 电话语音、视频、直播等】！！
3.5 TCP可能是1对1， UDP【多了1对多。没有多对多？？？】支持1对1 和 1对多。
3.6 TCP首部有20字节【为啥要那么多字节，为了记录 网路、拥塞情况、进行流量控制？？】，UDP只有8字节。

4 WebSocket的实现和应用。
4.1 websocket是html5中的协议。支持持久连接。http不支持持久连接。
http1.1 的keep-alive只是将请求合并成1个【不用来来回回的握手和挥手、首部也节省了？？ 那http2支持持久连接吗？？】。

4.2 websocket是基于【通过2首部 Connection、Upgrade 去完成 由http到websocket 的转换、升级的！！】http协议的。

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

5 HEAD的请求方式
与 GET请求类似，不过 返回的响应中 没有具体的内容，用户获取报头。
补充：options —— 允许客户端查看服务器的性能，比如说服务器支持的请求方式等。

6 几个很实用的BOM属性对象方法？？
6.1 location对象
location.href【url，不是uri。 URI包含的信息更全一些？？！】
location.search【查询字符串】
location.hash【# 符号之后的内容】
location.port
location.reload

6.2 history对象
history.go 可以是 history.go(num) num可取 -n ~ +n？？？
history.back 应该都是可以带上 num 参数的？？
historyforward

6.3 navigator对象
navigator.userAgent
navigator.cookieEnbled

7 介绍一下HTTP2.0【首部压缩、多路复用、服务端推送、二进制分帧、提升访问速度。】：
首部压缩。
允许多路复用。
服务端推送。
二进制分帧。
提升访问速度。

8 fetch发送2次请求的原因
fetch发送post【本质就是一次 options先询问是否支持修改的请求头。 第二次 “支持”，所以将 POST 改成 GET？？】。第一次状态码是 204【no content 无内容，注意 PUT DELETE 成功也是返回 204！！】。

原因很简单，因为你用fetch的post请求的时候，导致fetch 第一次发送了一个Options请求，询问服务器是否支持修改的请求头，如果服务器支持，则在第二次中发送真正的请求。

9 cookie、sessionStorage、localStorage
9.1 共同点：都是存于 B（浏览器）端，都是同源的！！

9.2 cookie 与 webstorage
cookie就是一个小文本，同源时、会自动在 B、S 之间传来传去，而webstorage（= session Storage + local Storage） 不会自动将数据发给服务器。
cookie有path的概念，可以限制cookie只属于某个路径下。
cookie大小只有 4k， 但是 webstorage 一般都是至少 5MB【不同浏览器的实现各不同】。
cookie的有效期是设置的（用户也可手动去删除），session是窗口关闭，local是得手动删除(不然会一直存在)！！

9.3 sessionStorage 和 localStorage
session 是当前窗口有效， locla是始终有效。
local【和cookie】是在所有同源的窗口中都是共享的。session好像即 同源的窗口 也不共享吧？？！

补充：cookie的作用【用户身份表示，记录用户选项、喜好等人性化！！！】：

保存用户登录状态。例如将用户id存储于一个cookie内，这样当用户下次访问该页面时就不需要重新登录了，现在很多论坛和社区都提供这样的功能。 cookie还可以设置过期时间，当超过时间期限后，cookie就会自动消失。因此，系统往往可以提示用户保持登录状态的时间：常见选项有一个月、三个 月、一年等。
跟踪用户行为。例如一个天气预报网站，能够根据用户选择的地区显示当地的天气情况。如果每次都需要选择所在地是烦琐的，当利用了cookie后就会显得很人性化了，系统能够记住上一次访问的地区，当下次再打开该页面时，它就会自动显示上次用户所在地区的天气情况。因为一切都是在后 台完成，所以这样的页面就像为某个用户所定制的一样，使用起来非常方便定制页面。如果网站提供了换肤或更换布局的功能，那么可以使用cookie来记录用户的选项，例如：背景色、分辨率等。当用户下次访问时，仍然可以保存上一次访问的界面风格。

10 webworker【独立于其他脚本，不阻塞主线程，通过 postMessage回传给主线程！！】
在HTML页面中，如果在执行脚本时，页面状态是不可响应的。

webworker是运行在后台的js，独立于其他脚本，不会影响页面的性能，通过postMessge将结果回传给 主线程！！！

10.1 创建 webworker：
检测 B端是否支持 webworker 。
创建 webworker文件 【js、回传函数等】
创建webworker对象

11 doctype 作用？？ 严格模式和混杂模式？？ HTML5不再基于 SGML(标准通用标记语言。) h5不用 进行 DTD 声明，因为它的标准只有一种！！！
doctype 声明位于文档最前面，告诉浏览器以何种方式进行渲染页面，这里有2种模式 —— 严格【以浏览器的最高标准去运行】 、 混杂【向后兼容，模拟老式浏览器】。

12 cookie【较简单，但是不好使，需要进行简单的封装！！！】如何防止XSS攻击
Set-cookie:
httponly 可以防止XSS，因为它会禁止 JS脚本去读取 cookie！！！
secure 告诉浏览器尽在请求为 https 时才发送 cookie.
结果应该是：

```
Set-Cookie=<cookie-value>.....
```

13 cookie 和 session 的区别【？？ 过于简短？！！】
HTTP是一个无状态协议，因此Cokkie的最大最永久时存储sessionId来表示用户【session应该也存了一份在服务端，过多的session存于服务端、性能会有问题应该？！！】

14 一句话概括 RESTFUL【过于简短，注意与 REST（一种架构？？）的区别？？】
用URL定位资源，用HTTP的请求方法描述操作（要对资源进行的动作），用 “消息体、报文实体” 传输需要的信息。
补充：查询一下 get、post、options、head、put（patch是新引入的，是局部更新的、put得传入完整的实体信息！！）、delete、connect、trace。

15 click在ios上有300ms 【为了区别 单纯的点击 和 缩放】延迟，原因以及如何解决？？
【通过 meta标签粗暴的 禁止用户缩放功能 —— user-scalable；
利用 fastclick， 点击后立刻模拟 click时间，把 300ms后的真正事件阻断掉。】
15.1 粗暴型，直接禁用缩放：
<mate name="viewport" content="width=device-width, user-scalable=no">

```
<meta name="viewport" content="width=device-width, user-scalable=no">
```

15.2 利用fastClick，原理：

监测到 touchend 事件后，立刻发 模拟click事件，并且把浏览器 300毫秒之后真正发的事件给阻断掉。

16 addEventListener 参数 【三个，event、function、useCaptrue 去指定时间是否在 捕获或 冒泡阶段执行。】

二 HTTP/HTML/浏览器

1 介绍一下HTTP返回的各种状态码

总之，1xx是信息类，2xx是“成功类”，3xx是“不在这里”，4xx就是前端背锅，5xx就是后端背锅。

100 Continue 继续。客户端应继续其请求

101 Switching Protocols 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议

200 OK 请求成功。一般用于GET与POST请求

201 Created 已创建。成功请求并创建了新的资源

202 Accepted 已接受。已经接受请求，但未处理完成

203 Non-Authoritative Information 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本

204 No Content 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档

205 Reset Content 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域

206 Partial Content 部分内容。服务器成功处理了部分GET请求

300 Multiple Choices 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择

301 Moved Permanently 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替

302 Found 临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI

303 See Other 查看其它地址。与301类似。使用GET和POST请求查看

304 Not Modified 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源

305 Use Proxy 使用代理。所请求的资源必须通过代理访问

306 Unused 已经被废弃的HTTP状态码

307 Temporary Redirect 临时重定向。与302类似。使用GET请求重定向

400 Bad Request 客户端请求的语法错误，服务器无法理解

401 Unauthorized 请求要求用户的身份认证

402 Payment Required 保留，将来使用

403 Forbidden 服务器理解请求客户端的请求，但是拒绝执行此请求

404 Not Found 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面

405 Method Not Allowed 客户端请求中的方法被禁止

406 Not Acceptable 服务器无法根据客户端请求的内容特性完成请求

407 Proxy Authentication Required 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权

408 Request Time-out 服务器等待客户端发送的请求时间过长，超时

409 Conflict 服务器完成客户端的PUT请求是可能返回此代码，服务器处理请求时发生了冲突

410 Gone 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置

411 Length Required 服务器无法处理客户端发送的不带Content-Length的请求信息

412 Precondition Failed 客户端请求信息的先决条件错误

413 Request Entity Too Large 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息

414 Request-URI Too Large 请求的URI过长（URI通常为网址），服务器无法处理

415 Unsupported Media Type 服务器无法处理请求附带的媒体格式

416 Requested range not satisfiable 客户端请求的范围无效

417 Expectation Failed 服务器无法满足Expect的请求头信息

500 Internal Server Error 服务器内部错误，无法完成请求

501 Not Implemented 服务器不支持请求的功能，无法完成请求

502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

503 Service Unavailable 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中

504 Gateway Time-out 充当网关或代理的服务器，未及时从远端服务器获取请求

505 HTTP Version not supported 服务器不支持请求的HTTP协议的版本，无法完成处理

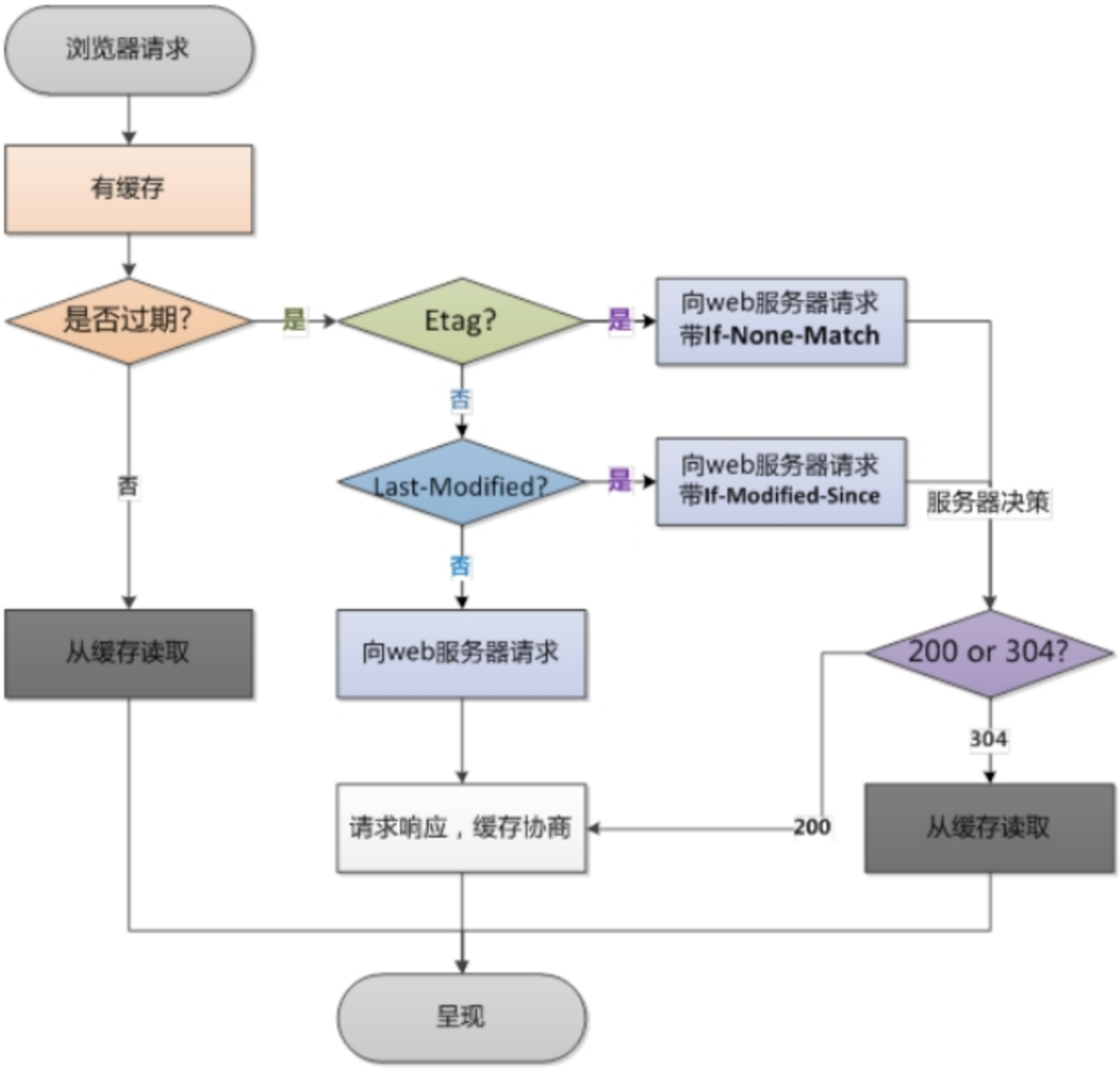
2 2种缓存 —— 强缓存（cache-control > expires） 和 协商缓存（etag > last-modified）

	获取资源形式	状态码	发送请求到服务器
强缓存	从缓存取	200（from cache）	否，直接从缓存取
协商缓存	从缓存取	304（not modified）	是，通过服务器来告知缓存是否可用

强缓存相关字段有 expires、cache-control【优先级最高】。
协商缓存相关的字段有 Etag/If-None-Match【优先级最高】、Last-Modified/If-Modifified-Since.

3 强缓存、协商缓存使用时期？？！

服务器上的资源不是一尘不变的，大多数情况下我会更新，如果我们还访问本地缓存，那么对用户来说就相当于资源没有更新、不符合预期！！！
综上，我们希望当资源真的有所更新时就去请求服务器上的最新资源并更新浏览器本地的资源。如果没有更新就使用本地的缓存、以最大程度的减少因网络请求而产生的资源浪费。
【发一个附带条件的请求才 300B 左右，如果索要请求的资源很大，如 3M ，那么协商缓存策略是很有价值、性价比的！！！】



4 前端优化【应该来一次大整合！！！ 点、线、网！！】
1 降低请求量： 合并资源，减少HTTP请求数，minify/gzip压缩，webP、lazyLoad。

2 加速请求速度：预解析DNS,减少域名数【？？？ 啥原理。有专门放置资源的域名，这样这些域名无需传送cookie了】，并行加载，CDN【2大核心：缓存、回源】。

3 缓存：HTTP协议缓存请求【即附带条件请求 --> 304】，离线化manifest，离线数据缓存

localStorage。
渲染：JS/CSS优化，加载顺序，服务端渲染【Django就是后端渲染？？ 解决 React页面应用 首屏加载慢的 问题？？】，pipeline。

4 GET 和 POST 对比 【2者本质都是TCP链接。
GET是幂等的，
但 POST不是幂等的！！】。
4.0 同： GET 和 POST 本质上都是TCP链接，并无差别。但是由于HTTP的规定、浏览器|服务器 等的限制，导致它们过程中有些不同！！

4.1 语义不同，一般来说，GET是请求资源、POST多用于增、改资源等。

4.2 GET产生一个TCP数据包，POST产生2个TCP数据包【多了1个 Options 预请求。有人经过抓包分析、好像不是？？？】。

4.3 get参数放于URL，POST放于request body中。

4.4 get请求的参数是放在url，浏览器对url长度有限制【其实http协议、标准对url长度没有进行限制！！是浏览器、客户端、服务端的“做的手脚”】，而 post的 request body 没有长度等限制！！！

4.5 get比post更不安全，因为参数直接暴露在url中，所以get不可以用来传递敏感的信息【get的参数信息放于url，可以带参数的存放到书签中，但是post不能带参的存放于书签中！！！】

4.6 get只能进行url编码，而post支持多种编码方式。

4.7 get请求、浏览器会主动 cache【post呢？？？ cache是啥？？？】

5 301【1很直，所以是永久重定向】 和 302【2很弯，所以是临时重定向】 区别【一般，301用于域名的跳转，302用于未登录用户访问用户信息中心时、临时重定向到登录页面！！】

301 Moved Permanently 被请求的资源已永久移动到新位置，并且将来任何对此资源的引用都应该使用本响应返回的若干个URI之一。如果可能，拥有链接编辑功能的客户端应当自动把请求的地址修改为从服务器反馈回来的地址。除非额外指定，否则这个响应也是可缓存的。

302 Found 请求的资源现在临时从不同的URI响应请求。由于这样的重定向是临时的，客户端应当继续向原有地址发送以后的请求。只有在Cache-Control或Expires中进行了指定的情况下，这个响应才是可缓存的。

字面上的区别就是301是永久重定向，而302是临时重定向。

301比较常用的场景是使用域名跳转。302用来做临时跳转 比如未登陆的用户访问用户中心重定向到登录页面。

6 HTTP支持的方法列表：
GET、POST、HEAD、OPTIONS、PUT、DELETE、TRACE(”诊断回路“)、CONNECT【后2者做啥的？？】

7 HTML5新增的元素【webStorage 是H5新增的！！！】
7.1 为了提供更好的web语义化，增加了 各种语义化标签 —— header、footer、nav、

aside、section.

7.2 表单方面，为了增强表单，为 input 增加了color、email、data、range等类型【抽空好好试一试，顺便查查各浏览器的兼容性！！】

7.3 在存储方面，提供了sessionStorage、localStorage和离线存储。

7.4 多媒体方面，新增了 audio 和 vido。

7.5 加了定理定位、画布canvas、拖放 drag、多线程编程的webworker【独立于主线程的、后台运行的JS脚本吧？？！】、全双工的websocket。

8 在地址栏输入一个URL后，到这个页面呈现出来，中间发生了什么？？

输入url后，首先需要找到这个url域名的服务器ip,为了寻找这个ip，浏览器首先会寻找缓存，查看缓存中是否有记录，缓存的查找记录为：浏览器缓存-》系统缓存-》路由器缓存，缓存中没有则查找系统的hosts文件中是否有记录，如果没有则查询DNS服务器，得到服务器的ip地址后，浏览器根据这个ip以及相应的端口号，构造一个http请求，这个请求报文会包括这次请求的信息，主要是请求方法，请求说明和请求附带的数据，并将这个http请求封装在一个tcp包中，这个tcp包会依次经过传输层，网络层，数据链路层，物理层到达服务器，服务器解析这个请求来作出响应，返回相应的html给浏览器，因为html是一个树形结构，浏览器根据这个html来构建DOM树，在dom树的构建过程中如果遇到JS脚本和外部JS连接，则会停止构建DOM树来执行和下载相应的代码，这会造成阻塞，这就是为什么推荐JS代码应该放在html代码的后面，之后根据外部央视，内部央视，内联样式构建一个CSS对象模型树CSSOM树，构建完成后和DOM树合并为渲染树，这里主要做的是排除非视觉节点，比如script，meta标签和排除display为none的节点，之后进行布局，布局主要是确定各个元素的位置和尺寸，之后是渲染页面，因为html文件中会含有图片，视频，音频等资源，在解析DOM的过程中，遇到这些都会进行并行下载，浏览器对每个域的并行下载数量有一定的限制，一般是4-6个，当然在这些所有的请求中我们还需要关注的就是缓存，缓存一般通过Cache-Control、Last-Modify、Expires等首部字段控制。 Cache-Control和Expires的区别在于Cache-Control使用相对时间，Expires使用的是基于服务器 端的绝对时间，因为存在时差问题，一般采用Cache-Control，在请求这些有设置了缓存的数据时，会先 查看是否过期，如果没有过期则直接使用本地缓存，过期则请求并在服务器校验文件是否修改，如果上一次 响应设置了ETag值会在这次请求的时候作为If-None-Match的值交给服务器校验，如果一致，继续校验 Last-Modified，没有设置ETag则直接验证Last-Modified，再决定是否返回304

9 cookie（携带于http头部中。）和session(存于服务端，可存放于 文件、数据库、内存中。)的区别； localStorage 和 sessionStorage 的区别。

Cookie和session都可用来存储用户信息，cookie存放于客户端，session存放于服务器端，因为cookie存放于客户端有可能被窃取，所以cookie一般用来存放不敏感的信息，比如用户设置的网站主题，敏感的信息用session存储，比如用户的登陆信息，session可以存放于文件，数据库，内存中都可以，cookie可以服务器端响应的时候设置，也可以客户端通过JS设置cookie会在请求时在http首部发送给客户端，cookie一般在客户端有大小限制，一般为4K，

下面从几个方向区分一下cookie，localstorage，sessionstorage的区别

1、生命周期：

Cookie：可设置失效时间，否则默认为关闭浏览器后失效

Localstorage:除非被手动清除，否则永久保存

Sessionstorage：仅在当前网页会话下有效，关闭页面或浏览器后就会被清除

2、存放数据：

Cookie：4k左右

Localstorage和sessionstorage：可以保存5M的信息

3、http请求：

Cookie：每次都会携带在http头中，如果使用cookie保存过多数据会带来性能问题

其他两个：仅在客户端即浏览器中保存，不参与和服务器的通信

4、易用性：

Cookie：需要程序员自己封装，原生的cookie接口不友好

其他两个：即可采用原生接口，亦可再次封装

5、应用场景：

从安全性来说，因为每次http请求都回携带cookie信息，这样子浪费了带宽，所以cookie应该尽可能的少用，此外cookie还需要指定作用域，不可以跨域调用，限制很多，但是用户识别用户登陆来说，cookie还是比storage好用，其他情况下可以用storage，localstorage可以用来在页面传递参数，sessionstorage可以用来保存一些临时的数据，防止用户刷新页面后丢失了一些参数，

10 常见的HTTP首部

10.1 可以将HTTP首部分为通用首部、请求首部、响应首部、实体首部【应该还有一个 其他

首部？？！】。

10.2 通用首部表示一些通用的信息，如 date表示报文创建时间。

10.3 请求首部就是请求报文中独有的，如 cookie 和 缓存相关【附带条件请求】的如 Etag、If-Modified-Since

10.4 响应首部就是响应报文中独有的，如 set-cookie 和重定向相关的 location。

10.5 实体首部用来描述实体部分，如 alow用来描述可执行的的请求方法，content-type描述主体（实体）类型，content-encoding描述主体的编码方式。

11 HTTP2.0 新的特性：
11.1 内容安全，因为http/2.0 是基于https【但2.0比https性能更高】的，天然具有安全的特性，通过http2.0可以避免单纯使用https的性能下降。

11.2 二进制格式。

2、二进制格式，http1.X的解析是基于文本的，http2.0将所有的传输信息分割为更小的消息和帧，并对他们采用二进制格式编码，基于二进制可以让协议有更多的扩展性，比如引入了帧来传输数据和指令

11.3 多路复用【相当于keep-alive长连接的增强版。还引入了流的优先级概念】。

3、多路复用，这个功能相当于是长连接的增强，每个request请求可以随机的混杂在一起，接收方可以根据request的id将request再归属到各自不同的服务端请求里面，另外多路复用中也支持了流的优先级，允许客户端告诉服务器那些内容是更优先级的资源，可以优先传输，

12 cache-control（单向的？）的取值范围。
cache-control是一个通用的消息头部被用于HTTP请求和响应中。
通过该指令实现缓存机制，是单向的！！！【那 meta标签可以设置它？？？ 啥意思？？】

常见的取值范围有：private、no-store【更绝情的】、no-cache、max-age【s-max-age好像是对代理服务器生效的？？！】、must-revalidate等。默认为 private 【意思是？？】！！！

13 浏览器生成页面的时候，会生成哪2棵树【DOM、CSSOM】

构造两棵树，DOM树和CSSOM规则树

当浏览器接收到服务器相应来的HTML文档后，会遍历文档节点，生成DOM树，

CSSOM规则树由浏览器解析CSS文件生成，

注意：CSSOM规则树应该是解析多个、所有的CSS文件而生成的吧？？！

三 HTTP/HTML/浏览器
1 XSS 和 CSRF 网络攻击以及 防范
1.1 XSS: 跨站脚本攻击。攻击者通过注入恶意的脚本，在用户浏览网页时进行攻击，如 获取 cookie ，或者其他用户身份信息。

可以分为 存储型 和 反射型。
存储型是 攻击者输入一些数据并存储到数据库中，其他浏览者看到的时候进行攻击。【啥意思？？？】
反射型的话 不存储在数据库中， 往往表现为 将攻击代码放在 URL地址的请求 参数中，防御的话为 cookie 设置 httpOnly 属性，对用户的输入进行检查、进行特殊字符的过滤【转成 字符实体， 如 > 是 &gt？？】

大汇总：

XSS, 即为（Cross Site Scripting），中文名为跨站脚本, 是发生在目标用户的浏览器层面上的，当渲染DOM树的过程成发生了不在预期内执行的JS代码时，就发生了XSS攻击。大多数XSS攻击的主要方式是嵌入一段远程或者第三方域上的JS代码。实际上是在目标网站的作用域下执行了这段js代码。

CSRF（Cross Site Request Forgery，跨站请求伪造），字面理解意思就是在别的站点伪造了一个请求。专业术语来说就是在受害者访问一个网站时，其 Cookie 还没有过期的情况下，攻击者伪造一个链接地址发送受害者并欺骗让其点击，从而形成 CSRF 攻击。

XSS防御的总体思路是：对输入(和URL参数)进行过滤，对输出进行编码。也就是对提交的所有内容进行过滤，对url中的参数进行过滤，过滤掉会导致脚本执行的相关内容；然后对动态输出到页面的内容进行html编码，使脚本无法在浏览器中执行。虽然对输入过滤可以被绕过，但是也还是会拦截很大一部分的XSS攻击。

防御CSRF 攻击主要有三种策略：验证 HTTP Referer 字段；在请求地址中添加 token 并验证；在 HTTP 头中自定义属性并验证。

1.2 CSRF: 跨站请求伪造。可以理解为 攻击者 盗用了用户的身份、以用户的名义发送了恶意的请求。

CSRF：跨站请求伪造，可以理解为攻击者盗用了用户的身份，以用户的名义发送了恶意请求，比如用户登录了一个网站后，立刻在另一个ｔａｂ页面访问量攻击者用来制造攻击的网站，这个网站要求访问刚刚登陆的网站，并发送了一个恶意请求，这时候CSRF就产生了，比如这个制造攻击的网站使用一张图片，但是这种图片的链接却是可以修改数据库的，这时候攻击者就可以以用户的名义操作这个数据库，防御方式的话：使用验证码，检查https头部的refer，使用token

防御方式：
使用验证码，生物特征识别如指纹、人脸识别，检查 https 头部的refer【甚至可以加上 自定义的 HTTP头进行对比分析！！！】，使用 token【token在服务器生成？？？ 其生成的算法？？】

2 怎么查看网站的性能？？【被动去测 + 主动监测】
2.1检测网页加载时间一般有2种方式：被动去测 + 主动监测。

2.2 被动检测：就是在被检测的页面置入脚本、探针，当用户访问网页时，探针自动采集数据并传回进行分析，
主动检测：主动地搭建分布式受控环境，模拟用户发请求，主动采集数据、以进行性能分析。
可以使用 第三方工具，如 性能极客； 网站就是 webpageTest。

3 输入URL并回车 到页面的加载显示 发生了什么?
DNS解析得到ip、TCP的3次握手连接、发送HTTP请求、服务器处理请求并返回HTTP报文、浏览器解析得到DOM和CSSOM树去合成 render tree 渲染页面、连接结束。

4 cookie 和 session 的认识。 cookie的限制？？
4.1 cookie放于 客户端C、被携带于 http头部来回传送，session 放于服务器端S。

4.2 cookie不安全，别人可以分析存放在本地的cookie、进行cookie欺骗【CSRF攻击？？】，考虑到安全应该使用 session？！！

4.3 session是放于服务端的，当访问增多、服务器性能可能会下降，这时应考虑下 cookie的使用 【混合、结合使用？？】。

4.4 单个cookie保存的数据不可超过4k，很多浏览器都限制一个站点最多 存放 20个cookie。

5 cookie可以设置的的字段：
name cookie名称；
value 一个cookie的值；
domain 可以访问改cookie的域名【可多个域名？？ 类似于 cookie的权限控制了吧？？！】
汇总：

name字段为一个cookie的名称。

value字段为一个cookie的值。

domain字段为可以访问此cookie的域名。

非顶级域名，如二级域名或者三级域名，设置的cookie的domain只能为顶级域名或者二级域名或者三级域名本身，不能设置其他二级域名的cookie，否则cookie无法生成。

顶级域名只能设置domain为顶级域名，不能设置为二级域名或者三级域名，否则cookie无法生成。

二级域名能读取设置了domain为顶级域名或者自身的cookie，不能读取其他二级域名domain的cookie。所以要想cookie在多个二级域名中共享，需要设置domain为顶级域名，这样就可以在所有二级域名里面或者到这个cookie的值了。
顶级域名只能获取到domain设置为顶级域名的cookie，其他domain设置为二级域名的无法获取。

path字段为可以访问此cookie的页面路径。 比如domain是abc.com,path是/test，那么只有/test路径下的页面可以读取此cookie。

expires/Max-Age 字段为此cookie超时时间。若设置其值为一个时间，那么当到达此时间后，此cookie失效。不设置的话默认值是Session，意思是cookie会和session一起失效。当浏览器关闭(不是浏览器标签页，而是整个浏览器) 后，此cookie失效。

Size字段 此cookie大小。

http字段 cookie的httponly属性。若此属性为true，则只有在http请求头中会带有此cookie的信息，而不能通过document.cookie来访问此cookie。

secure 字段 设置是否只能通过https来传递此条cookie

补充：
cookie的编码格式： encodeURI()

6 H5C3的新东西说一说呗？？
6.1 html5:
标签的增删：
新增的语义元素，header、footer、section、aside、nav、main、acticle、figure【图片相关的好像？？？】
内容元素，mark【高亮。可以试试、然后兼容性 —— CanIUse】、progress【进度】。
新的表单控件 calendar、date、time、email、url、search【带搜索框的？？？】
新的input类型 color、date、datetime、email等。
移除的：big、font、frame、frameset。

多媒体： audio video source【前面2个的资源列表标签元素 应该】 embed track。

canvas画布：支持内联SVG、支持 MathML。

本地离线存储【原理 与 实践？？】： 把需要离线存储在本地的文件列在 manifest 配置文

件。

web存储【即 webStorage】： localStorage、sessionStorage。

6.2 CSS3:
边框：border-radius、box-shadow。
背景：background-size、background-origin。
2、3D转换，如 transform【参数个数不同而区分开 2、3D？？？】
动画 animation 【注意与transition 区别开来。动画的 “过程帧” 更多、更细腻？？！！】

7 web性能优化：
降低请求量： 合并资源【如精灵图】，减少HTTP请求数【长连接 Connection: keep-alive】，minify/gzip压缩，webP、lazyLoad。
加快请求速度：预解析DNS【meta标签的 DNS-prefetch？？】，减少域名数【啥原理？？减少3次握手的次数？？】，并行加载【如 link 标签，而不是 @import 方式引入 CSS样式】，CDN分发。

缓存：HTTP协议缓存请求【附带条件请求， 304 Not Modified】，离线缓存manifest【将要缓存在本地的文件列表 写入 manifest文件中？？】，离线数据缓存 localStorage。

渲染：JS/CSS优化【压缩等手段，写法”标准化、最小化“？？】，加载顺序，服务端渲染，pipeline【管道？？】

完