1. 跨标签页通讯？

不同标签页间的通讯，本质原理就是去运用一些可以 共享的中间介质，因此比较常用的有以下方法：

- 通过父页面 `window.open()`和子页面 `postMessage`
  - 异步下，通过 `window.open('about: blank')`和 `tab.location.href = '*'`
- 设置同域下共享的 `localStorage` 与监听 `window.onstorage`
  - 重复写入相同的值无法触发
  - 会受到浏览器隐身模式等的限制
- 设置共享 `cookie` 与不断轮询脏检查(`setInterval`)
- 借助服务端或者中间层实现

2. 浏览器架构？

- 用户界面
- 主进程
- 内核
  - 渲染引擎
  - `JS` 引擎
    - 执行栈
- 事件触发线程
  - 消息队列
    - 微任务
    - 宏任务
- 网络异步线程
- 定时器线程

3. 浏览器下事件循环(Event Loop)？

事件循环是指: 执行一个宏任务，然后执行清空微任务列表，循环再执行宏任务，再清微任务列表

- 微任务 `microtask(jobs): promise / ajax / Object.observe` (该方法已废弃)
- 宏任务 `macrotask(task): setTimout / script / IO / UI Rendering`

4. 重绘与回流？

当元素的样式发生变化时，浏览器需要触发更新，重新绘制元素。这个过程中，有两种类型的操作，即重绘与回流微绘。

- 重绘(repaint): 当元素样式的改变不影响布局时，浏览器将使用重绘对元素进行更新，此时由于只需要 UI 层面的重新像素绘制，因此 损耗较少
- 回流(reflow): 当元素的尺寸、结构或触发某些属性时，浏览器会重新渲染页面，称为回流。此时，浏览器需要重新经过计算，计算后还需要重新页面布局，因此是较重的操作。会触发回流的操作:
- 页面初次渲染
- 浏览器窗口大小改变
- 元素尺寸、位置、内容发生改变
- 元素字体大小变化
- 添加或者删除可见的 dom 元素
- 激活 CSS 伪类（例如：:hover ）
- 查询某些属性或调用某些方法
  - clientWidth、clientHeight、clientTop、clientLeft
  - offsetWidth、offsetHeight、offsetTop、offsetLeft
  - scrollWidth、scrollHeight、scrollTop、scrollLeft
  - getComputedStyle()
  - getBoundingClientRect()
  - scrollTo()

注意:回流必定触发重绘，重绘不一定触发回流。重绘的开销较小，回流的代价较高

最佳实践:

- CSS
  - 避免使用 table 布局
  - 将动画效果应用到 position 属性为 absolute 或 fixed 的元素上
- javascript
  - 避免频繁操作样式，可汇总后统一 一次修改
  - 尽量使用 class 进行样式修改
  - 减少 dom 的增删次数，可使用 字符串 或者 documentFragment 一次性插入
  - 极限优化时，修改样式可将其 display: none 后修改
  - 避免多次触发上面提到的那些会触发回流的方法，可以的话尽量用 变量存住

5. 内存泄露？

- 意外的全局变量: 无法被回收
- 定时器: 未被正确关闭，导致所引用的外部变量无法被释放
- 事件监听: 没有正确销毁 (低版本浏览器可能出现)
- 闭包: 会导致父级中的变量无法被释放

- dom 引用:dom 元素被删除时，内存中的引用未被正确清空
- 可用 chrome 中的 timeline 进行内存标记，可视化查看内存的变化情况，找出异常点

6. 前端需要注意哪些 SEO？

- 合理的 title、description、keywords：搜索对着三项的权重逐个减小，titl 值强调重点即可，重要关键词出现不要超过 2 次，而且要靠前，不同页面 title 要有所不同 description 把页面内容高度概括，长度合适，不可过分堆砌关键词，不同页面 description 有所不同；keywords 列举出重要关键词即可。
- 语义化的 HTML 代码，符合 W3C 规范：语义化代码让搜索引擎容易理解网页
- 重要内容 HTML 代码放在最前：搜索引擎抓取 HTML 顺序是从上到下，有的搜索引擎对抓取长度有限制，保证重要内容一定会被抓取
- 重要内容不要用 js 输出：爬虫不会执行 js 获取内容
- 少用 iframe ：搜索引擎不会抓取 iframe 中的内容
- 非装饰性图片必须加 alt
- 提高网站速度：网站速度是搜索引擎排序的一个重要指标

7. HTTP 的几种请求方法用途？

- GET 方法
  - 发送一个请求来取得服务器上的某一资源
- POST 方法
  - 向 URL 指定的资源提交数据或附加新的数据
- PUT 方法
  - 跟 POST 方法很像，也是想服务器提交数据。但是，它们之间有不同。PUT 指定了资源在服务器上的位置，而 POST 没有
- HEAD 方法
  - 只请求页面的首部
- DELETE 方法
  - 删除服务器上的某资源
- OPTIONS 方法

- 它用于获取当前 URL 所支持的方法。如果请求成功，会有一个 Allow 的头包含类似 “GET,POST” 这样的信息

- TRACE 方法
  - TRACE 方法被用于激发一个远程的，应用层的请求消息回路
- CONNECT 方法
  - 把请求连接转换到透明的 TCP/IP 通道

8. 从浏览器地址栏输入 url 到显示页面的步骤？

基础版本

- 浏览器根据请求的 URL 交给 DNS 域名解析，找到真实 IP ，向服务器发起请求；
- 服务器交给后台处理完成后返回数据，浏览器接收文件（HTML、JS、CSS 图象等）；
- 浏览器对加载到的资源（HTML、JS、CSS 等）进行语法解析，建立相应的内部数据结构（如 HTML 的 DOM ）；
- 载入解析到的资源文件，渲染页面，完成。

详细版

1.在浏览器地址栏输入 URL

2. 浏览器查看**缓存**，如果请求资源在缓存中并且新鲜，跳转到转码步骤
   1. 如果资源未缓存，发起新请求
   2. 如果已缓存，检验是否足够新鲜，足够新鲜直接提供给客户端，否则与服务器进行验证。
   3. 检验新鲜通常有两个 HTTP 头进行控制 Expires 和 Cache-Control ：
      - HTTP1.0 提供 Expires，值为一个绝对时间表示缓存新鲜日期
      - HTTP1.1 增加了 Cache-Control: max-age=,值为以秒为单位的最大新鲜时间
3. 浏览器**解析** URL 获取协议，主机，端口，path

4.浏览器**组装一个** HTTP（GET）**请求报文**

5. 浏览器**获取主机 ip 地址**，过程如下：
   1. 浏览器缓存
   2. 本机缓存
   3. hosts 文件
   4. 路由器缓存
   5. ISP DNS 缓存
   6. DNS 递归查询（可能存在负载均衡导致每次 IP 不一样）

6. **打开一个 socket 与目标 IP 地址，端口建立 TCP 链接**，三次握手如下：
    1. 客户端发送一个 TCP 的 SYN=1，Seq=X 的包到服务器端口
    2.服务器发回 SYN=1， ACK=X+1， Seq=Y 的响应包
    3. 客户端发送 ACK=Y+1， Seq=Z
7. TCP 链接建立后**发送 HTTP 请求**
8. 服务器接受请求并解析，将请求转发到服务程序，如虚拟主机使用 HTTP Host 头部判断请求的服务程序
9. 服务器检查 HTTP **请求头是否包含缓存验证信息**如果验证缓存新鲜，返回 304 等对应状态码
10. 处理程序读取完整请求并准备 HTTP 响应，可能需要查询数据库等操作
11. 服务器将**响应报文通过 TCP 连接发送回浏览器**
12. 浏览器接收 HTTP 响应，然后根据情况选择**关闭 TCP 连接或者保留重用，关闭 TCP 连接四次握手如下**：
    1. 主动方发送 Fin=1， Ack=Z， Seq= X 报文
    2. 被动方发送 ACK=X+1， Seq=Z 报文
    3. 被动方发送 Fin=1， ACK=X， Seq=Y 报文
    4. 主动方发送 ACK=Y， Seq=X 报文
13.浏览器检查响应状态吗：是否为 1XX，3XX， 4XX， 5XX，这些情况处理与 2XX 不同
14. 如果资源可缓存，**进行缓存**
15. 对响应进行**解码**（例如 gzip 压缩）
16. 根据资源类型决定如何处理（假设资源为 HTML 文档）
17. **解析 HTML 文档，构件 DOM 树，下载资源，构造 CSSOM 树，执行 js 脚本**，这些操作没严格的先后顺序，以下分别解释
18. **构建 DOM 树**：
    1. Tokenizing：根据 HTML 规范将字符流解析为标记
    2. Lexing：词法分析将标记转换为对象并定义属性和规则
    3. DOM construction：根据 HTML 标记关系将对象组成 DOM 树
19. 解析过程中遇到图片、样式表、js 文件，**启动下载**
20. 构建 CSSOM **树**：
    1. Tokenizing：字符流转换为标记流
    2. Node：根据标记创建节点
    3. CSSOM：节点创建 CSSOM 树
21. 根据 DOM 树和 CSSOM 树构建渲染树 ：
    1. 从 DOM 树的根节点遍历所有**可见节点**，不可见节点包括：1） `script` , `meta` 这样身不可见的标签。2)被 css 隐藏的节点，如 `display: none`
    2. 对每一个可见节点，找到恰当的 CSSOM 规则并应用
    3.发布可视节点的内容和计算样式

22. js **解析如下：**
    1. 浏览器创建 Document 对象并解析 HTML，将解析到的元素和文本节点添加到文档中此时 document.readystate **为** loading
    2. HTML 解析器遇到**没有** async **和** defer **的** script **时**，将他们添加到文档中，然后执行行或外部脚本。这些脚本会同步执行，并且在脚本下载和执行时解析器会暂停。这样就可以用 document.write()把文本插入到输入流中。**同步脚本经常简单定义函数和注册事件处理程序，他们可以遍历和操作** script **和他们之前的文档内容**
    3. 当解析器遇到设置了 async 属性的 script 时，开始下载脚本并继续解析文档。脚本会在它**下载完成后尽快执行**，但是**解析器不会停下来等它下载**。异步脚本**禁止使用** document.write()，它们可以访问自己 script 和之前的文档元素
    4. 当文档完成解析，document.readState 变成 interactive
    5. 所有 defer 脚本会**按照在文档出现的顺序执行**，延迟脚本**能访问完整文档树**，禁止使用 document.write()
    6. 浏览器**在** Document **对象上触发** DOMContentLoaded **事件**
    7. 此时文档完全解析完成，浏览器可能还在等待如图片等内容加载，等这些**内容完成载入并且所有异步脚本完成载入和执行**，document.readState 变为 complete，window 触发 load 事件
23. **显示页面**（HTML 解析过程中会逐步显示页面）

9. 如何进行网站性能优化？

- `content` 方面
    - 减少 `HTTP` 请求：合并文件、`CSS` 精灵、`inline Image`
    - 减少 `DNS` 查询：`DNS` 缓存、将资源分布到恰当数量的主机名
    - 减少 `DOM` 元素数量
- `Server` 方面
    - 使用 `CDN`
    - 配置 `Etag`
    - 对组件使用 `Gzip` 压缩
- `Cookie` 方面
    - 减小 `cookie` 大小
- `css` 方面
    - 将样式表放到页面顶部
    - 不使用 `CSS` 表达式
    - 使用 `<link>` 不使用 `@import`

- Javascript 方面
  - 将脚本放到页面底部
  - 将 javascript 和 css 从外部引入
  - 压缩 javascript 和 css
  - 删除不需要的脚本
  - 减少 DOM 访问
- 图片方面
  - 优化图片：根据实际颜色需要选择色深、压缩
  - 优化 css 精灵
  - 不要在 HTML 中拉伸图片

10. HTTP 状态码及其含义？

- 1XX ： 信息状态码
  - 100 Continue 继续，一般在发送 post 请求时，已发送了 http header 之后服务端将返回此信息，表示确认，之后发送具体参数信息
- 2XX ： 成功状态码
  - 200 OK 正常返回信息
  - 201 Created 请求成功并且服务器创建了新的资源
  - 202 Accepted 服务器已接受请求，但尚未处理
- 3XX ： 重定向
  - 301 Moved Permanently 请求的网页已永久移动到新位置。
  - 302 Found 临时性重定向。
  - 303 See Other 临时性重定向，且总是使用 GET 请求新的 URI 。
  - 304 Not Modified 自从上次请求后，请求的网页未修改过。
- 4XX ： 客户端错误
  - 400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。
  - 401 Unauthorized 请求未授权。
  - 403 Forbidden 禁止访问
  - 404 Not Found 找不到如何与 URI 相匹配的资源。
- 5XX: 服务器错误
  - 500 Internal Server Error 最常见的服务器端错误。
  - 503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）

11. 介绍一下你对浏览器内核的理解？

- 主要分成两部分：渲染引擎(layout engineer 或 Rendering Engine)和 JS 引擎
- 渲染引擎：负责取得网页的内容（HTML 、XML 、图像等等）、整理讯息（例如加入 CSS 等），以及计算网页的显示方式，然后会输出至显示器或打印机。浏览器的内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其它需要编辑、显示网络内容的应用程序都需要内核
- JS 引擎则：解析和执行 javascript 来实现网页的动态效果
- 最开始渲染引擎和 JS 引擎并没有区分的很明确，后来 JS 引擎越来越独立，内核就倾向于只指渲染引擎

12. 浏览器是怎么对 HTML5 的离线储存资源进行管理和加载的呢？

- 在线的情况下，浏览器发现 html 头部有 manifest 属性，它会请求 manifest 文件，如果是第一次访问 app ，那么浏览器就会根据 manifest 文件的内容下载相应的资源并且进行离线存储。如果已经访问过 app 并且资源已经离线存储了，那么浏览器就会使用离线的资源加载页面，然后浏览器会对比新的 manifest 文件与旧的 manifest 文件，如果文件没有发生改变，就不做任何操作，如果文件改变了，那么就会重新下载文件中的资源并进行离线存储
- 离线的情况下，浏览器就直接使用离线存储的资源

13. 网页验证码是干嘛的，是为了解决什么安全问题？

- 区分用户是计算机还是人的公共全自动程序。可以防止恶意破解密码、刷票、论坛灌水
- 有效防止黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试

14. 为什么利用多个域名来存储网站资源会更有效？

- CDN 缓存更方便
- 突破浏览器并发限制
- 节约 cookie 带宽
- 节约主域名的连接数，优化页面响应速度
- 防止不必要的安全问题

15. HTTP request 报文结构是怎样的？

- 首行是 Request-Line 包括：**请求方法**，**请求** URI，**协议版本**，CRLF
- 首行之后是若干行**请求头**，包括 general-header，request-header 或者 entity-header，每个一行以 CRLF 结束
- 请求头和消息实体之间有一个 CRLF **分隔**

- 根据实际请求需要可能包含一个**消息实体** 一个请求报文例子如下：

```
GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1
Host: www.w3.org
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML
Referer: https://www.google.com.hk
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: authorstyle=yes
If-None-Match: "2cc8-3e3073913b100
If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT
```

16. HTTP response 报文结构是怎样的？

- 首行是状态行包括：HTTP **版本，状态码，状态描述**，后面跟一个 CRLF
- 首行之后是**若干行响应头**，包括：**通用头部，响应头部，实体头部**
- 响应头部和响应实体之间用**一个 CRLF 空行**分隔
- 最后是一个可能的**消息实体** 响应报文例子如下：

17. 浏览器存储？

- 短暂性的时候，我们只需要将数据存在内存中，只在运行时可用
- 持久性存储，可以分为 浏览器端 与 服务器端
  - 浏览器:
    - `cookie` ：通常用于存储用户身份，登录状态等
      `http` 中自动携带， 体积上限为 `4K` ， 可自行设置过期时间
    - `localStorage / sessionStorage` ：长久储存/窗口关闭删除， 体积限制为 `4~5M`
    - `indexDB`
  - 服务器:
    - 分布式缓存 `redis`
    - 数据库

18. Web Worker？

- 方面现代浏览器为 `JavaScript` 创造的 多线程环境。可以新建并将部分任务分配到 `worker` 线程并行运行，两个线程可 独立运行，互不干扰，可通过自带的消息机制 相互通信
- `Websocket` 是一个 持久化的协议， 基于 `http` ， 服务端可以 主动 `push`

- 基本用法

```
// 创建 worker         js
const worker = new Worker('work.js');
// 向主进程推送消息
worker.postMessage('Hello World');
// 监听主进程来的消息
worker.onmessage = function (event) {
    console.log('Received message ' + event.data);
}
```

19. http 缓存策略:可分为强缓存和协商缓存？

- `Cache-Control / Expires`：浏览器判断缓存是否过期，未过期时，直接使用强缓存，`Cache-Control` 的 `max-age` 优先级高于 `Expires`
- 当缓存已经过期时，使用协商缓存
  - 唯一标识方案: `Etag`（`response` 携带) &`If-None-Match` (`request` 携带，上一次返回的 `Etag`): 服务器判断资源是否被修改
  - 最后一次修改时间: `Last-Modified(response) & If-Modified-Since`（`request`，上一次返回的 `Last-Modified` ）
    - 如果一致，则直接返回 304 通知浏览器使用缓存
    - 如不一致，则服务端返回新的资源
- `Last-Modified` 缺点：
  - 周期性修改，但内容未变时，会导致缓存失效
  - 最小粒度只到 `s`，`s` 以内的改动无法检测到
- `Etag` 的优先级高于 `Last-Modified`

20. get / post？

- `get`：缓存、请求长度受限、会被历史保存记录
  - 无副作用(不修改资源)，幂等(请求次数与资源无关)的场景
- `post`：安全、大数据、更多编码类型

21. TCP 三次握手？

建立连接前，客户端和服务端需要通过握手来确认对方:

- 客户端发送 `syn` (同步序列编号) 请求，进入 `syn_send` 状态，等待确认
- 服务端接收并确认 `syn` 包后发送 `syn+ack` 包，进入 `syn_recv` 状态
- 客户端接收 `syn+ack` 包后，发送 `ack` 包，双方进入 `established` 状态

22. TCP 四次挥手？

- 客户端 -- FIN --> 服务端， FIN—WAIT
- 服务端 -- ACK --> 客户端， CLOSE-WAIT
- 服务端 -- ACK,FIN --> 客户端， LAST-ACK
- 客户端 -- ACK --> 服务端，CLOSED

22. 如何解决跨域问题？

事件先了解下浏览器的同源策略 同源策略 /SOP（Same origin policy） 是一种约定，由 Netscape 公司 1995 年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到 XSS、 CSFR 等攻击。所谓同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个 ip 地址，也非同源

- 通过 jsonp 跨域
- 通 nginx 代理跨域
- 通 nodejs 中间件代理跨域
- 通后端在头部信息里面设置安全域名

23. Node 的 Event Loop: 6 个阶段？

- timer 阶段: 执行到期的 setTimeout / setInterval 队列回调
- I/O 阶段: 执行上轮循环残流的 callback
- idle , prepare
- poll ：等待回调
  - 执行回调
  - 执行定时器
    - 如有到期的 setTimeout / setInterval ，则返回 timer 阶段
    - 如有 setImmediate ，则前往 check 阶段
- check
  - 执行 setImmediate
- close callbacks

24. 安全性问题？

- XSS 攻击: 注入恶意代码
  - cookie 设置 httpOnly
  - 转义页面上的输入内容和输出内容
- CSRF ：跨站请求伪造，防护:
  - get 不修改数据

- 不被第三方网站访问到用户的 cookie
- 设置白名单，不被第三方网站请求
- 请求校验

25. 常见 web 安全及防护原理？

- sql 注入原理
  - 就是通过把 SQL 命令插入到 Web 表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令
- 总的来说有以下几点
  - 永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等
  - 永远不要使用动态拼装 SQL，可以使用参数化的 SQL 或者直接使用存储过程进行数据查询存取
  - 永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接
  - 不要把机密信息明文存放，请加密或者 hash 掉密码和敏感的信息

SS 原理及防范

- Xss(cross-site scripting) 攻击指的是攻击者往 Web 页面里插入恶意 html 标签或者 javascript 代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取 cookie 中的用户私密信息；或者攻击者在论坛中加一个恶意表单，当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点

XSS 防范方法

- 首先代码里对用户输入的地方和变量都需要仔细检查长度和对 ”<”,”>”,”;”,”’” 等字符做过滤；其次任何内容写到页面之前都必须加以 encode，避免不小心把 html tag 弄出来。这一个层面做好，至少可以堵住超过一半的 XSS 攻击

XSS 与 CSRF 有什么区别吗？

- XSS 是获取信息，不需要提前知道其他用户页面的代码和数据包。 CSRF 是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。要完成一次 CSRF 攻击，受害者必须依次完成两个步骤
- 登录受信任网站 A ，并在本地生成 Cookie
- 在不登出 A 的情况下，访问危险网站 B

CSRF 的防御

- 服务端的 CSRF 方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数
- 通过验证码的方法

26. 常见兼容性问题？

- png24 位的图片在 iE6 浏览器上出现背景，解决方案是做成 PNG8
- 浏览器默认的 margin 和 padding 不同。解决方案是加一个全局的*{margin:0;padding:0;}来统一，但是全局效率很低，一般是如下这样解决：

```
body,ul,li,ol,dl,dt,dd,form,input,h1,h2,h3,h4,h5,h6,p p{
    margin:0;
    padding:0;
}
```

- IE 下，event 对象有 x ，y 属性,但是没有 pageX ，pageY 属性
- Firefox 下，event 对象有 pageX，pageY 属性,但是没有 x,y 属性