HTML、HTTP、web综合问题

1. 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，`title`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

2 . `<img>`的`title`和`alt`有什么区别?

- 通常当鼠标滑动到元素上的时候显示
- `alt`是`<img>`的特有属性，是图片内容的等价描述，用于图片无法加载时显示、读屏器阅读图片。可提图片高可访问性，除了纯装饰图片外都必须设置有意义的值，搜索引擎会重点分析。

3. 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

4. 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连接的四次握手如下：

    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**，这些操作没有严格的先后顺序，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`这样本身不可见的标签。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，然后执行行内或外部脚本。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）

详细简版

1. 从浏览器接收`url`到开启网络请求线程（这一部分可以展开浏览器的机制以及进程与线程之间的关系）
2. 开启网络线程到发出一个完整的`HTTP`请求（这一部分涉及到dns查询，`TCP/IP`请求，五层因特网协议栈等知识）
3. 从服务器接收到请求到对应后台接收到请求（这一部分可能涉及到负载均衡，安全拦截以及后台内部的处理等等）
4. 后台和前台的`HTTP`交互（这一部分包括`HTTP`头部、响应码、报文结构、`cookie`等知识，可以提下静态资源的`cookie`优化，以及编码解码，如`gzip`压缩等）
5. 单独拎出来的缓存问题，`HTTP`的缓存（这部分包括http缓存头部，`ETag`，`catch-control`等）
6. 浏览器接收到`HTTP`数据包后的解析流程（解析`html`-词法分析然后解析成`dom`树、解析`css`生成`css`规则树、合并成`render`树，然后`layout`、`painting`渲

染、复合图层的合成、GPU绘制、外链资源的处理、loaded和DOMContentLoaded等）
7. CSS的可视化格式模型（元素的渲染规则，如包含块，控制框，BFC，IFC等概念）
8. JS引擎解析过程（JS的解释阶段，预处理阶段，执行阶段生成执行上下文，VO，作用域链、回收机制等等）
9. 其它（可以拓展不同的知识模块，如跨域，web安全，hybrid模式等等内容）

5. 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

你有用过哪些前端性能优化的方法?

- 减少http请求次数：CSS Sprites, JS、CSS源码压缩、图片大小控制合适；网页Gzip，CDN托管，data缓存 ，图片服务器。

- 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存AJAX请求结果，每次操作本地变量，不用请求，减少请求次数
- 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能。
- 当需要设置的样式很多时设置className而不是直接操作style
- 少用全局变量、缓存DOM节点查找的结果。减少IO读取操作
- 避免使用CSS Expression（css表达式)又称Dynamic properties(动态属性)
- 图片预加载，将样式表放在顶部，将脚本放在底部 加上时间戳
- 避免在页面的主体布局中使用table，table要等其中的内容完全下载之后才会显示出来，显示比div+css布局慢

谈谈性能优化问题

- 代码层面：避免使用css表达式，避免使用高级选择器，通配选择器
- 缓存利用：缓存Ajax，使用CDN，使用外部js和css文件以便缓存，添加Expires头，服务端配置Etag，减少DNS查找等
- 请求数量：合并样式和脚本，使用css图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载
- 请求带宽：压缩文件，开启GZIP

前端性能优化最佳实践?

- 性能评级工具（PageSpeed 或 YSlow）
- 合理设置 HTTP 缓存：Expires 与 Cache-control
- 静态资源打包，开启 Gzip 压缩（节省响应流量）
- CSS3 模拟图像，图标base64（降低请求数）
- 模块延迟(defer)加载/异步(async)加载
- Cookie 隔离（节省请求流量）
- localStorage（本地存储）
- 使用 CDN 加速（访问最近服务器）
- 启用 HTTP/2（多路复用，并行加载）
- 前端自动化（gulp/webpack）

6. 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）。

7. 语义化的理解

- 用正确的标签做正确的事情!
- `HTML`语义化就是让页面的内容结构化，便于对浏览器、搜索引擎解析;
- 在没有样式`CSS`情况下也以一种文档格式显示，并且是容易阅读的。
- 搜索引擎的爬虫依赖于标记来确定上下文和各个关键字的权重，利于 `SEO`。
- 使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解

8. 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

常见的浏览器内核有哪些

- `Trident`内核：`IE,MaxThon,TT,The World,360`,搜狗浏览器等。[又称MSHTML]
- `Gecko`内核：`Netscape6`及以上版本，`FF,MozillaSuite/SeaMonkey`等
- `Presto`内核：`Opera7`及以上。[`Opera`内核原为：Presto，现为：`Blink`;]

- `Webkit`内核：`Safari,Chrome`等。[`Chrome`的`Blink`（`WebKit`的分支）]

9. html5有哪些新特性、移除了那些元素?

- `HTML5` 现在已经不是 `SGML` 的子集，主要是关于图像，位置，存储，多任务等功能的增加
  - 新增选择器 `document.querySelector`、`document.querySelectorAll`
  - 拖拽释放(`Drag and drop`) API
  - 媒体播放的 `video` 和 `audio`
  - 本地存储 `localStorage` 和 `sessionStorage`
  - 离线应用 `manifest`
  - 桌面通知 `Notifications`
  - 语意化标签 `article`、`footer`、`header`、`nav`、`section`
  - 增强表单控件 `calendar`、`date`、`time`、`email`、`url`、`search`
  - 地理位置 `Geolocation`
  - 多任务 `webworker`
  - 全双工通信协议 `websocket`
  - 历史管理 `history`
  - 跨域资源共享(CORS) `Access-Control-Allow-Origin`
  - 页面可见性改变事件 `visibilitychange`
  - 跨窗口通信 `PostMessage`
  - `Form Data` 对象
  - 绘画 `canvas`
- 移除的元素：
  - 纯表现的元素：`basefont`、`big`、`center`、`font`、 `s`、`strike`、`tt`、`u`
  - 对可用性产生负面影响的元素：`frame`、`frameset`、`noframes`
- 支持`HTML5`新标签：
  - `IE8/IE7/IE6`支持通过`document.createElement`方法产生的标签
  - 可以利用这一特性让这些浏览器支持`HTML5`新标签
  - 浏览器支持新标签后，还需要添加标签默认的样式
- 当然也可以直接使用成熟的框架、比如`html5shim`

如何区分 HTML 和 HTML5

- `DOCTYPE`声明、新增的结构元素、功能元素

10. HTML5的离线储存怎么使用，工作原理能不能解释一下?

- 在用户没有与因特网连接时，可以正常访问站点或应用，在用户与因特网连接时，更新用户机器上的缓存文件
- 原理：HTML5的离线存储是基于一个新建的.appcache文件的缓存机制(不是存储技术)，通过这个文件上的解析清单离线存储资源，这些资源就会像cookie一样被存储了下来。之后当网络在处于离线状态下时，浏览器会通过被离线存储的数据进行页面展示
- 如何使用：
  - 页面头部像下面一样加入一个manifest的属性；
  - 在cache.manifest文件的编写离线存储的资源
  - 在离线状态时，操作window.applicationCache进行需求实现

```
CACHE MANIFEST
#v0.11
CACHE:
js/app.js
css/style.css
NETWORK:
resourse/logo.png
FALLBACK:
/offline.html
```

11. 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。

12. 请描述一下 `cookies`，`sessionStorage` 和 `localStorage` 的区别?

- `cookie`是网站为了标示用户身份而储存在用户本地终端（Client Side）上的数据（通常经过加密）
- `cookie`数据始终在同源的http请求中携带（即使不需要），记会在浏览器和服务器间来回传递
- `sessionStorage`和`localStorage`不会自动把数据发给服务器，仅在本地保存
- 存储大小：
  - `cookie`数据大小不能超过4k
  - `sessionStorage`和`localStorage`虽然也有存储大小的限制，但比`cookie`大得多，可以达到5M或更大
- 有期时间：
  - `localStorage` 存储持久数据，浏览器关闭后数据不丢失除非主动删除数据
  - `sessionStorage` 数据在当前浏览器窗口关闭后自动删除
  - `cookie` 设置的`cookie`过期时间之前一直有效，即使窗口或浏览器关闭

13. iframe有那些缺点?

- `iframe`会阻塞主页面的`Onload`事件
- 搜索引擎的检索程序无法解读这种页面，不利于`SEO`
- `iframe`和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载
- 使用`iframe`之前需要考虑这两个缺点。如果需要使用`iframe`，最好是通过`javascript`动态给`iframe`添加`src`属性值，这样可以绕开以上两个问题

14. WEB标准以及W3C标准是什么?

- 标签闭合、标签小写、不乱嵌套、使用外链`css`和`js`、结构行为表现的分离

15. xhtml和html有什么区别?

- 一个是功能上的差别
  - 主要是`XHTML`可兼容各大浏览器、手机以及`PDA`，并且浏览器也能快速正确地编译网页
- 另外是书写习惯的差别
  - `XHTML` 元素必须被正确地嵌套，闭合，区分大小写，文档必须拥有根元素

16. Doctype作用? 严格模式与混杂模式如何区分？它们有何意义?

- 页面被加载的时，`link`会同时被加载，而`@imort`页面被加载的时，`link`会同时被加载，而`@import`引用的`CSS`会等到页面被加载完再加载 `import`只在`IE5`以上才能识别，而`link`是`XHTML`标签，无兼容问题 `link`方式的样式的权重 高于`@import`的权重
- `<!DOCTYPE>` 声明位于文档中的最前面，处于 `<html>` 标签之前。告知浏览器的解析器， 用什么文档类型 规范来解析这个文档
- 严格模式的排版和 `JS` 运作模式是 以该浏览器支持的最高标准运行
- 在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。 `DOCTYPE`不存在或格式不正确会导致文档以混杂模式呈现

17. 行内元素有哪些？块级元素有哪些？ 空(void)元素有那些？行内元素和块级元素有什么区别?

- 行内元素有：`a b span img input select strong`
- 块级元素有：`div ul ol li dl dt dd h1 h2 h3 h4… p`
- 空元素：`<br> <hr> <img> <input> <link> <meta>`
- 行内元素不可以设置宽高，不独占一行
- 块级元素可以设置宽高，独占一行

18. HTML全局属性(global attribute)有哪些

- `class`:为元素设置类标识
- `data-*`: 为元素增加自定义属性
- `draggable`: 设置元素是否可拖拽
- `id`: 元素`id`，文档内唯一
- `lang`: 元素内容的的语言
- `style`: 行内`css`样式
- `title`: 元素相关的建议信息

19. Canvas和SVG有什么区别?

- `svg`绘制出来的每一个图形的元素都是独立的`DOM`节点，能够方便的绑定事件或用来修改。`canvas`输出的是一整幅画布
- `svg`输出的图形是矢量图形，后期可以修改参数来自由放大缩小，不会失真和锯齿。而`canvas`输出标量画布，就像一张图片一样，放大会失真或者锯齿

20. HTML5 为什么只需要写 <!DOCTYPE HTML>

- `HTML5` 不基于 `SGML`，因此不需要对`DTD`进行引用，但是需要`doctype`来规范浏览器的行为
- 而`HTML4.01`基于`SGML`,所以需要对`DTD`进行引用，才能告知浏览器文档所使用的文档类型

21. 如何在页面上实现一个圆形的可点击区域?

- `svg`
- `border-radius`
- 纯`js`实现 需要求一个点在不在圆上简单算法、获取鼠标坐标等等

22. 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

23. viewport

```
 <meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    // width     设置viewport宽度，为一个正整数，或字符串‘device-width’
    // device-width   设备宽度
    // height    设置viewport高度，一般设置了宽度，会自动解析出高度，可以不用设置
    // initial-scale     默认缩放比例（初始缩放比例），为一个数字，可以带小数
    // minimum-scale     允许用户最小缩放比例，为一个数字，可以带小数
    // maximum-scale     允许用户最大缩放比例，为一个数字，可以带小数
    // user-scalable     是否允许手动缩放
```

- 延伸提问
    - 怎样处理 移动端 `1px` 被 渲染成 `2px`问题

局部处理

- `meta`标签中的 `viewport`属性， `initial-scale` 设置为 `1`
- `rem`按照设计稿标准走，外加利用`transfrome` 的`scale(0.5)` 缩小一倍即可；

全局处理

- `mate`标签中的`viewport`属性，`initial-scale`设置为`0.5`
- `rem`按照设计稿标准走即可

24. 渲染优化

- 禁止使用`iframe`（阻塞父文档`onload`事件）
  - `iframe`会阻塞主页面的`Onload`事件
  - 搜索引擎的检索程序无法解读这种页面，不利于SEO
  - `iframe`和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载
  - 使用`iframe`之前需要考虑这两个缺点。如果需要使用`iframe`，最好是通过`javascript`
  - 动态给`iframe`添加`src`属性值，这样可以绕开以上两个问题
- 禁止使用`gif`图片实现`loading`效果（降低`CPU`消耗，提升渲染性能）
- 使用`CSS3`代码代替`JS`动画（尽可能避免重绘重排以及回流）
- 对于一些小图标，可以使用base64位编码，以减少网络请求。但不建议大图使用，比较耗费`CPU`
  - 小图标优势在于
    - 减少`HTTP`请求
    - 避免文件跨域
    - 修改及时生效
- 页面头部的`<style></style>` `<script></script>`会阻塞页面；（因为`Renderer`进程中`JS`线程和渲染线程是互斥的）
- 页面中空的`href`和`src`会阻塞页面其他资源的加载(阻塞下载进程)
- 网页`gzip`，`CDN`托管，`data`缓存，图片服务器
- 前端模板JS+数据，减少由于`HTML`标签导致的带宽浪费，前端用变量保存AJAX请求结果，每次操作本地变量，不用请求，减少请求次数
- 用`innerHTML`代替`DOM`操作，减少`DOM`操作次数，优化`javascript`性能
- 当需要设置的样式很多时设置`className`而不是直接操作`style`
- 少用全局变量、缓存`DOM`节点查找的结果。减少`IO`读取操作
- 图片预加载，将样式表放在顶部，将脚本放在底部 加上时间戳
- 对普通的网站有一个统一的思路，就是尽量向前端优化、减少数据库操作、减少磁盘`IO`

25. meta viewport相关

```
<!DOCTYPE html>  <!--H5标准声明，使用 HTML5 doctype，不区分大小写-->
<head lang="en"> <!--标准的 lang 属性写法-->
<meta charset='utf-8'>    <!--声明文档使用的字符编码-->
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>   <!--优先使用 IE 最新版本和 Chrome-->
<meta name="description" content="不超过150个字符"/>        <!--页面描述-->
<meta name="keywords" content=""/>     <!-- 页面关键词-->
<meta name="author" content="name, email@gmail.com"/>    <!--网页作者-->
<meta name="robots" content="index,follow"/>       <!--搜索引擎抓取-->
<meta name="viewport" content="initial-scale=1, maximum-scale=3, minimum-scale=1, user-scalable=no"> <!--为移动设备添加viewport-->
<meta name="apple-mobile-web-app-title" content="标题"> <!--iOS 设备 begin-->
<meta name="apple-mobile-web-app-capable" content="yes"/>  <!--添加到主屏后的标题（iOS 6 新增）
是否启用 WebApp 全屏模式，删除苹果默认的工具栏和菜单栏-->
<meta name="apple-itunes-app" content="app-id=myAppStoreID, affiliate-data=myAffiliateData, app-argument=myURL">
<!--添加智能 App 广告条 Smart App Banner（iOS 6+ Safari）-->
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
<meta name="format-detection" content="telphone=no, email=no"/>  <!--设置苹果工具栏颜色-->
<meta name="renderer" content="webkit"> <!-- 启用360浏览器的极速模式(webkit)-->
<meta http-equiv="X-UA-Compatible" content="IE=edge">     <!--避免IE使用兼容模式-->
<meta http-equiv="Cache-Control" content="no-siteapp" />
 <!--不让百度转码-->
<meta name="HandheldFriendly" content="true">     <!--针对手持设备优化，主要是针对一些老的不识别viewport的浏览器，比如黑莓-->
<meta name="MobileOptimized" content="320">    <!--微软的老式浏览器-->
<meta name="screen-orientation" content="portrait">    <!--uc强制竖屏-->
```

```
<meta name="x5-orientation" content="portrait">    <!--QQ强制竖屏-->
<meta name="full-screen" content="yes">              <!--UC强制全屏-->
<meta name="x5-fullscreen" content="true">       <!--QQ强制全屏-->
<meta name="browsermode" content="application">   <!--UC应用模式-->
<meta name="x5-page-mode" content="app">   <!-- QQ应用模式-->
<meta name="msapplication-tap-highlight" content="no">    <!--windows phone 点击无高亮
设置页面不缓存-->
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="expires" content="0"><!DOCTYPE html>  <!--H5标准声明，使用 HTML5 doctype，不区分大小写-->
<head lang="en"> <!--标准的 lang 属性写法-->
<meta charset='utf-8'>    <!--声明文档使用的字符编码-->
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"/>   <!--优先使用 IE 最新版本和 Chrome-->
<meta name="description" content="不超过150个字符"/>       <!--页面描述-->
<meta name="keywords" content=""/>     <!-- 页面关键词-->
<meta name="author" content="name, email@gmail.com"/>    <!--网页作者-->
<meta name="robots" content="index,follow"/>      <!--搜索引擎抓取-->
<meta name="viewport" content="initial-scale=1, maximum-scale=3, minimum-scale=1, user-scalable=no"> <!--为移动设备添加viewport-->
<meta name="apple-mobile-web-app-title" content="标题"> <!--iOS 设备 begin-->
<meta name="apple-mobile-web-app-capable" content="yes"/>  <!--添加到主屏后的标题（iOS 6 新增）
是否启用 WebApp 全屏模式，删除苹果默认的工具栏和菜单栏-->
<meta name="apple-itunes-app" content="app-id=myAppStoreID, affiliate-data=myAffiliateData, app-argument=myURL">
<!--添加智能 App 广告条 Smart App Banner（iOS 6+ Safari）-->
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
<meta name="format-detection" content="telphone=no, email=no"/>  <!--设置苹果工具栏颜色-->
<meta name="renderer" content="webkit"> <!-- 启用360浏览器的极速模式(webkit)-->
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">      <!--避免IE使用兼容模式-->
<meta http-equiv="Cache-Control" content="no-siteapp" />  <!--不让百度转码-->
<meta name="HandheldFriendly" content="true">      <!--针对手持设备优化，主要是针对一些老的不识别viewport的浏览器，比如黑莓-->
<meta name="MobileOptimized" content="320">    <!--微软的老式浏览器-->
<meta name="screen-orientation" content="portrait">    <!--uc强制竖屏-->
<meta name="x5-orientation" content="portrait">     <!--QQ强制竖屏-->
<meta name="full-screen" content="yes">                <!--UC强制全屏-->
<meta name="x5-fullscreen" content="true">         <!--QQ强制全屏-->
<meta name="browsermode" content="application">    <!--UC应用模式-->
<meta name="x5-page-mode" content="app">    <!-- QQ应用模式-->
<meta name="msapplication-tap-highlight" content="no">     <!--windows phone 点击无高亮
设置页面不缓存-->
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="expires" content="0">
```

26. 你做的页面在哪些流览器测试过？这些浏览器的内核分别是什么?

- `IE`: `trident`内核
- `Firefox`： `gecko`内核
- `Safari`:`webkit`内核
- `Opera`:以前是`presto`内核，`Opera`现已改用Google - `Chrome`的`Blink`内核
- `Chrome:Blink`(基于`webkit`，Google与Opera Software共同开发)

27. div+css的布局较table布局有什么优点?

- 改版的时候更方便 只要改`css`文件。
- 页面加载速度更快、结构化清晰、页面显示简洁。
- 表现与结构相分离。
- 易于优化（`seo`）搜索引擎更友好，排名更容易靠前。

28. a：img的alt与title有何异同？b：strong与em的异同?

- `alt(alt text)`:为不能显示图像、窗体或`applets`的用户代理（`UA`），`alt`属性用来指定替换文字。替换文字的语言由`lang`属性指定。(在IE浏览器下会在没有`title`时把`alt`当成`tool tip`显示)
- `title(tool tip)`:该属性为设置该属性的元素提供建议性的信息
- `strong`:粗体强调标签，强调，表示内容的重要性
- `em`:斜体强调标签，更强烈强调，表示内容的强调点

29. 你能描述一下渐进增强和优雅降级之间的不同吗

- 渐进增强：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进和追加功能达到更好的用户体验。
- 优雅降级：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

> 区别：优雅降级是从复杂的现状开始，并试图减少用户体验的供给，而渐进增强则是从一个非常基础的，能够起作用的版本开始，并不断扩充，以适应未来环境的需要。降级（功能衰减）意味着往回看；而渐进增强则意味着朝前看，同时保证其根基处于安全地带

30. 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

31. 简述一下src与href的区别

- `src`用于替换当前元素，href用于在当前文档和引用资源之间确立联系。
- `src`是`source`的缩写，指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求`src`资源时会将其指向的资源下载并应用到文档内，例如`js`脚本，`img`图片和`frame`等元素

> <script src ="js.js"> 当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部

- `href`是`Hypertext Reference`的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果我们在文档中添加

- `<link href="common.css" rel="stylesheet"/>`那么浏览器会识别该文档为`css`文件，就会并行下载资源并且不会停止对当前文档的处理。这也是为什么建议使用`link`方式来加载`css`，而不是使用`@import`方式

32. 知道的网页制作会用到的图片格式有哪些?

- `png-8`、`png-24`、`jpeg`、`gif`、`svg`

> 但是上面的那些都不是面试官想要的最后答案。面试官希望听到是`Webp`,`Apng`。（是否有关注新技术，新鲜事物）

- **Webp**：`WebP`格式，谷歌（google）开发的一种旨在加快图片加载速度的图片格式。图片压缩体积大约只有`JPEG`的`2/3`，并能节省大量的服务器带宽资源和数据空间。`Facebook Ebay`等知名网站已经开始测试并使用`WebP`格式。
- 在质量相同的情况下，WebP格式图像的体积要比JPEG格式图像小`40%`。
- **Apng**：全称是“`Animated Portable Network Graphics`”,是PNG的位图动画扩展，可以实现png格式的动态图片效果。04年诞生，但一直得不到各大浏览器厂商的支持，直到日前得到`iOS safari 8`的支持，有望代替`GIF`成为下一代动态图标准

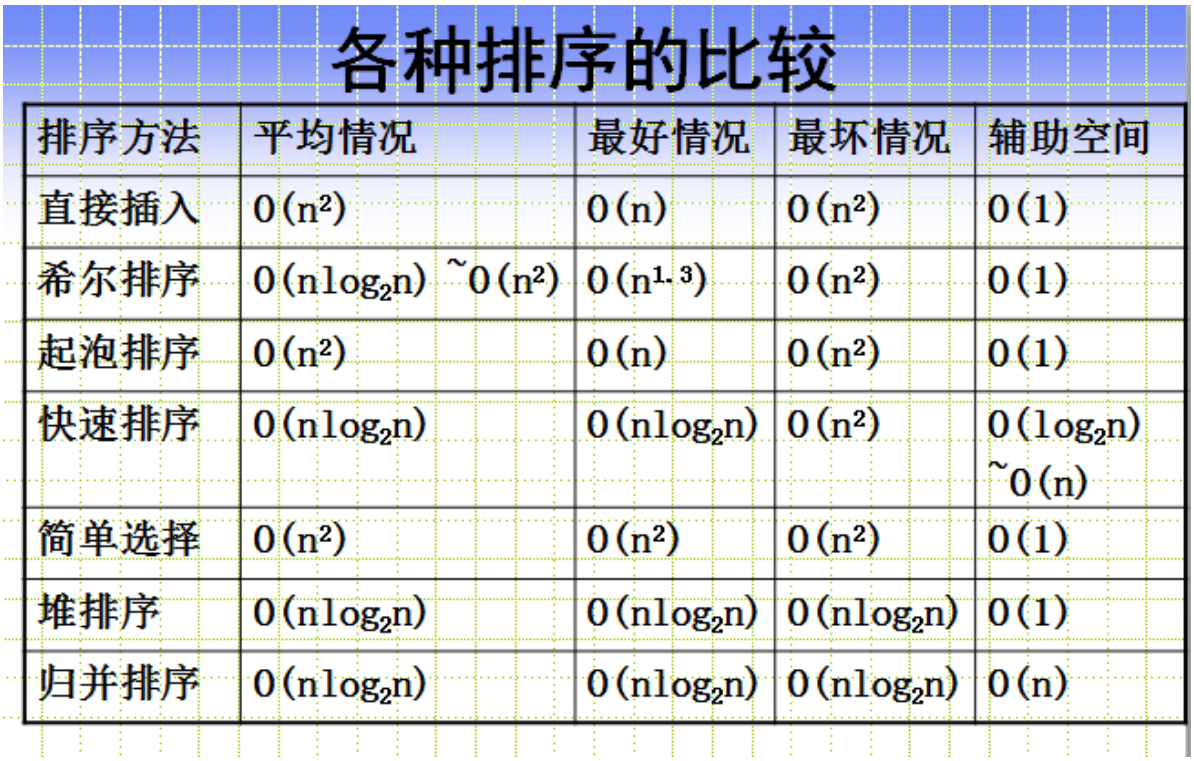
33. 在css/js代码上线之后开发人员经常会优化性能，从用户刷新网页开始，一次js请求一般情况下有哪些地方会有缓存处理?

> `dns`缓存，`cdn`缓存，浏览器缓存，服务器缓存

33. 一个页面上有大量的图片（大型电商网站），加载很慢，你有哪些方法优化这些图片的加载，给用户更好的体验。

- 图片懒加载，在页面上的未可视区域可以添加一个滚动事件，判断图片位置与浏览器顶端的距离与页面的距离，如果前者小于后者，优先加载。
- 如果为幻灯片、相册等，可以使用图片预加载技术，将当前展示图片的前一张和后一张优先下载。
- 如果图片为css图片，可以使用`CSSsprite`，`SVGsprite`，`Iconfont`、`Base64`等技术。
- 如果图片过大，可以使用特殊编码的图片，加载时会先加载一张压缩的特别厉害的缩略图，以提高用户体验。
- 如果图片展示区域小于图片的真实大小，则因在服务器端根据业务需要先行进行图片压缩，图片压缩后大小与展示一致。

34. 常见排序算法的时间复杂度,空间复杂度



各种排序的比较

排序方法	平均情况	最好情况	最坏情况	辅助空间
直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
希尔排序	$O(n\log_2 n)$ ~$O(n^2)$	$O(n^{1.3})$	$O(n^2)$	$O(1)$
起泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$ ~$O(n)$
简单选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$

35. web开发中会话跟踪的方法有哪些

- `cookie`
- `session`
- `url`重写
- 隐藏`input`
- `ip`地址

36. HTTP request报文结构是怎样的

1. 首行是**Request-Line**包括：**请求方法**，**请求URI**，**协议版本**，**CRLF**
2. 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
3. 请求头和消息实体之间有一个**CRLF分隔**
4. 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=0.9,image/webp,*/*;q=0.8
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36
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

name=qiu&age=25
```

37. 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：

```
HTTP/1.1 200 OK
Date: Tue, 08 Jul 2014 05:28:43 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
ETag: "40d7-3e3073913b100"
Accept-Ranges: bytes
Content-Length: 16599
Cache-Control: max-age=21600
Expires: Tue, 08 Jul 2014 11:28:43 GMT
P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"
Content-Type: text/html; charset=iso-8859-1

{"name": "qiu", "age": 25}
```

38. title与h1的区别、b与strong的区别、i与em的区别

- `title`属性没有明确意义只表示是个标题，H1则表示层次明确的标题，对页面信息的抓取也有很大的影响
- `strong`是标明重点内容，有语气加强的含义，使用阅读设备阅读网络时：`<strong>`会重读，而`<B>`是展示强调内容
- i内容展示为斜体，em表示强调的文本

39. 请你谈谈Cookie的弊端

> `cookie`虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的

- 每个特定的域名下最多生成`20`个`cookie`
- `IE6`或更低版本最多`20`个`cookie`
- `IE7`和之后的版本最后可以有`50`个`cookie`
- `Firefox`最多50个`cookie`
- `chrome`和`Safari`没有做硬性限制
- IE 和 Opera 会清理近期最少使用的`cookie`，`Firefox`会随机清理`cookie`
- `cookie`的最大大约为`4096`字节，为了兼容性，一般设置不超过`4095`字节
- 如果`cookie`被人拦截了，就可以取得所有的`session`信息

40. git fetch和git pull的区别

- `git pull`：相当于是从远程获取最新版本并`merge`到本地
- `git fetch`：相当于是从远程获取最新版本到本地，不会自动`merge`