HTML 面试知识点总结

本部分主要是笔者在复习 HTML 相关知识和一些相关面试题时所做的笔记，如果出现错误，希望大家指出！

目录

1. DOCTYPE 的作用是什么?

相关知识点：

> IE5.5 引入了文档模式的概念，而这个概念是通过使用文档类型（DOCTYPE）切换实现的。
>
> <!DOCTYPE>声明位于 HTML文档中的第一行，处于 <html> 标签之前。告知浏览器的解析器用什么文档标准解析这个文档。
>
> DOCTYPE 不存在或格式不正确会导致文档以兼容模式呈现。

回答（参考1-5）：

> <!DOCTYPE> 声明一般位于文档的第一行，它的作用主要是告诉浏览器以什么样的模式来解析文档。一般指定了之后会以标准模式来进行文档解析，否则就以兼容模式进行解析。在标准模式下，浏览器的解析规则都是按照最新的标准进行解析的。
> 而在兼容模式下，浏览器会以向后兼容的方式来模拟老式浏览器的行为，以保证一些老的网站的正确访问。
>
> 在 html5 之后不再需要指定 DTD 文档，因为 html5 以前的 html 文档都是基于 SGML 的，所以需要通过指定 DTD 来定义文档中允许的属性以及一些规则。而 html5 不再基于 SGML 了，所以不再需要使用 DTD。

2. 标准模式与兼容模式各有什么区别?

标准模式的渲染方式和 JS 引擎的解析方式都是以该浏览器支持的最高标准运行。在兼容模式中，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。

3. HTML5 为什么只需要写 <!DOCTYPE HTML>，而不需要引入 DTD?

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（让浏览器按照它们应该的方式来运
行）。

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。

4. SGML 、 HTML 、XML 和 XHTML 的区别?

SGML 是标准通用标记语言，是一种定义电子文档结构和描述其内容的国际标准语言，是所有电子文档标记语言的起源。

HTML 是超文本标记语言，主要是用于规定怎么显示网页。

XML 是可扩展标记语言是未来网页语言的发展方向，XML 和 HTML 的最大区别就在于 XML 的标签是可以自己创建的，数量无限多，
而 HTML 的标签都是固定的而且数量有限。

XHTML 也是现在基本上所有网页都在用的标记语言，他其实和 HTML 没什么本质的区别，标签都一样，用法也都一样，就是比 HTML
更严格，比如标签必须都用小写，标签都必须有闭合标签等。

5. DTD 介绍

DTD（ Document Type Definition 文档类型定义）是一组机器可读的规则，它们定义 XML 或 HTML 的特定版本中所有允许元
素及它们的属性和层次关系的定义。在解析网页时，浏览器将使用这些规则检查页面的有效性并且采取相应的措施。

DTD 是对 HTML 文档的声明，还会影响浏览器的渲染模式（工作模式）。

6. 行内元素定义

HTML4 中，元素被分成两大类: inline （内联元素）与 block（块级元素）。一个行内元素只占据它对应标签的边框所包含的空间。

常见的行内元素有 a b span img strong sub sup button input label select textarea

7. 块级元素定义

块级元素占据其父元素（容器）的整个宽度，因此创建了一个“块”。

常见的块级元素有 div ul ol li dl dt dd h1 h2 h3 h4 h5 h6 p

8. 行内元素与块级元素的区别?

HTML4中，元素被分成两大类：inline （内联元素）与 block （块级元素）。

（1） 格式上，默认情况下，行内元素不会以新行开始，而块级元素会新起一行。
（2） 内容上，默认情况下，行内元素只能包含文本和其他行内元素。而块级元素可以包含行内元素和其他块级元素。
（3） 行内元素与块级元素属性的不同，主要是盒模型属性上：行内元素不支持设置宽高；行内元素支持水平方向的padding、border、
margin；行内元素可以设置垂直方向的padding、border、margin，但是不会影响布局。

9. HTML5 元素的分类

HTML4中，元素被分成两大类: inline（内联元素）与 block（块级元素）。但在实际的开发过程中，因为页面表现的需要，前
端工程师经常把 inline 元素的 display 值设定为 block （比如 a 标签），也经常把 block 元素的 display 值设定为
inline 之后更是出现了 inline-block 这一对外呈现 inline 对内呈现 block 的属性。因此，简单地把 HTML 元素划分为
inline 与 block 已经不再符合实际需求。

HTML5中，元素主要分为7类：Metadata Flow Sectioning Heading Phrasing Embedded Interactive
Metadata（元数据型）：位于head中，决定其他内容的样式或行为。包括：<meta> <link> <title>等
Flow（文档流型）：大部分文档body内的元素
Sectioning（区块型）：定义区块内容范围的元素
Heading（标题型）：定义节的标题。包括<hgroup> <h1>-<h6>
Phrasing（语句型）：包含许多HTML4中的inline层次的元素。包括 <img> <sub> <sup>等
Embedded（内嵌型）：将其他资源导入（嵌入）到文档中。 包括<audio> <video> <canvas>等
Interactive（交互型）：专门用来与用户交互。包括<input> <a> <select>等

10. 空元素定义

标签内没有内容的 HTML 标签被称为空元素。空元素是在开始标签中关闭的。

常见的空元素有：br hr img input link meta

11. link 标签定义

link 标签定义文档与外部资源的关系。

link 元素是空元素，它仅包含属性。 此元素只能存在于 head 部分，不过它可出现任何次数。

link 标签中的 rel 属性定义了当前文档与被链接文档之间的关系。常见的 stylesheet 指的是定义一个外部加载的样式表。

12. 页面导入样式时，使用 link 和 @import 有什么区别?

（1）从属关系区别。 @import 是 CSS 提供的语法规则，只有导入样式表的作用；link 是 HTML 提供的标签，不仅可以加
载 CSS 文件，还可以定义 RSS、rel 连接属性、引入网站图标等。

（2）加载顺序区别。加载页面时，link 标签引入的 CSS 被同时加载；@import 引入的 CSS 将在页面加载完毕后被加载。

（3）兼容性区别。@import 是 CSS2.1 才有的语法，故只可在 IE5+ 才能识别；link 标签作为 HTML 元素，不存在兼容
性问题。

（4）DOM 可控性区别。可以通过 JS 操作 DOM ，插入 link 标签来改变样式；由于 DOM 方法是基于文档的，无法使用
@import 的方式插入样式。

13. 你对浏览器的理解?

浏览器的主要功能是将用户选择的 web 资源呈现出来，它需要从服务器请求资源，并将其显示在浏览器窗口中，资源的格式通常
是 HTML，也包括 PDF、image 及其他格式。用户用 URI（Uniform Resource Identifier 统一资源标识符）来指定所请
求资源的位置。

HTML 和 CSS 规范中规定了浏览器解释 html 文档的方式，由 W3C 组织对这些规范进行维护，W3C 是负责制定 web 标准的
组织。

但是浏览器厂商纷纷开发自己的扩展，对规范的遵循并不完善，这为 web 开发者带来了严重的兼容性问题。

简单来说浏览器可以分为两部分，shell 和 内核。

其中 shell 的种类相对比较多，内核则比较少。shell 是指浏览器的外壳：例如菜单，工具栏等。主要是提供给用户界面操作，
参数设置等等。它是调用内核来实现各种功能的。内核才是浏览器的核心。内核是基于标记语言显示内容的程序或模块。也有一些
浏览器并不区分外壳和内核。从 Mozilla 将 Gecko 独立出来后，才有了外壳和内核的明确划分。

14. 介绍一下你对浏览器内核的理解?

主要分成两部分：渲染引擎和 JS引擎。

渲染引擎的职责就是渲染，即在浏览器窗口中显示所请求的内容。默认情况下，渲染引擎可以显示 html、xml 文档及图片，它也
可以借助插件（一种浏览器扩展）显示其他类型数据，例如使用 PDF 阅读器插件，可以显示 PDF 格式。

JS 引擎：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。

15. 常见的浏览器内核比较

Trident：这种浏览器内核是 IE 浏览器用的内核，因为在早期 IE 占有大量的市场份额，所以这种内核比较流行，以前有很多
网页也是根据这个内核的标准来编写的，但是实际上这个内核对真正的网页标准支持不是很好。但是由于 IE 的高市场占有率，微
软也很长时间没有更新 Trident 内核，就导致了 Trident 内核和 W3C 标准脱节。还有就是 Trident 内核的大量 Bug 等
安全问题没有得到解决，加上一些专家学者公开自己认为 IE 浏览器不安全的观点，使很多用户开始转向其他浏览器。

Gecko：这是 Firefox 和 Flock 所采用的内核，这个内核的优点就是功能强大、丰富，可以支持很多复杂网页效果和浏览器扩
展接口，但是代价是也显而易见就是要消耗很多的资源，比如内存。

Presto：Opera 曾经采用的就是 Presto 内核，Presto 内核被称为公认的浏览网页速度最快的内核，这得益于它在开发时的
天生优势，在处理 JS 脚本等脚本语言时，会比其他的内核快3倍左右，缺点就是为了达到很快的速度而丢掉了一部分网页兼容性。

Webkit：Webkit 是 Safari 采用的内核，它的优点就是网页浏览速度较快，虽然不及 Presto 但是也胜于 Gecko 和 Trid
ent，缺点是对于网页代码的容错性不高，也就是说对网页代码的兼容性较低，会使一些编写不标准的网页无法正确显示。WebKit
前身是 KDE 小组的 KHTML 引擎，可以说 WebKit 是 KHTML 的一个开源的分支。

Blink：谷歌在 Chromium Blog 上发表博客，称将与苹果的开源浏览器核心 Webkit 分道扬镳，在 Chromium 项目中研发 B
link 渲染引擎（即浏览器核心），内置于 Chrome 浏览器之中。其实 Blink 引擎就是 Webkit 的一个分支，就像 webkit 是
KHTML 的分支一样。Blink 引擎现在是谷歌公司与 Opera Software 共同研发，上面提到过的，Opera 弃用了自己的 Presto
内核，加入 Google 阵营，跟随谷歌一起研发 Blink。

详细的资料可以参考：
《浏览器内核的解析和对比》
《五大主流浏览器内核的源起以及国内各大浏览器内核总结》

16. 常见浏览器所用内核

（1） IE 浏览器内核：Trident 内核，也是俗称的 IE 内核；

（2） Chrome 浏览器内核：统称为 Chromium 内核或 Chrome 内核，以前是 Webkit 内核，现在是 Blink内核；

（3） Firefox 浏览器内核：Gecko 内核，俗称 Firefox 内核；

（4） Safari 浏览器内核：Webkit 内核；

（5） Opera 浏览器内核：最初是自己的 Presto 内核，后来加入谷歌大军，从 Webkit 又到了 Blink 内核；

（6） 360浏览器、猎豹浏览器内核：IE + Chrome 双内核；

（7） 搜狗、遨游、QQ 浏览器内核：Trident（兼容模式）+ Webkit（高速模式）；

（8） 百度浏览器、世界之窗内核：IE 内核；

（9） 2345浏览器内核：好像以前是 IE 内核，现在也是 IE + Chrome 双内核了；

（10）UC 浏览器内核：这个众口不一，UC 说是他们自己研发的 U3 内核，但好像还是基于 Webkit 和 Trident ，还有说
是基于火狐内核。

17. 浏览器的渲染原理?

（1）首先解析收到的文档，根据文档定义构建一棵 DOM 树，DOM 树是由 DOM 元素及属性节点组成的。

（2）然后对 CSS 进行解析，生成 CSSOM 规则树。

（3）根据 DOM 树和 CSSOM 规则树构建渲染树。渲染树的节点被称为渲染对象，渲染对象是一个包含有颜色和大小等属性的矩
形，渲染对象和 DOM 元素相对应，但这种对应关系不是一对一的，不可见的 DOM 元素不会被插入渲染树。还有一些 DOM
元素对应几个可见对象，它们一般是一些具有复杂结构的元素，无法用一个矩形来描述。

（4）当渲染对象被创建并添加到树中，它们并没有位置和大小，所以当浏览器生成渲染树以后，就会根据渲染树来进行布局（也
可以叫做回流）。这一阶段浏览器要做的事情是要弄清楚各个节点在页面中的确切位置和大小。通常这一行为也被称为“自动
重排”。

（5）布局阶段结束后是绘制阶段，遍历渲染树并调用渲染对象的 paint 方法将它们的内容显示在屏幕上，绘制使用 UI 基础组
件。

值得注意的是，这个过程是逐步完成的，为了更好的用户体验，渲染引擎将会尽可能早的将内容呈现到屏幕上，并不会等到所有的
html 都解析完成之后再去构建和布局 render 树。它是解析完一部分内容就显示一部分内容，同时，可能还在通过网络下载其
余内容。

详细资料可以参考：
《浏览器渲染原理》
《浏览器的渲染原理简介》
《前端必读：浏览器内部工作原理》
《深入浅出浏览器渲染原理》

18. 渲染过程中遇到 JS 文件怎么处理？（浏览器解析过程）

JavaScript 的加载、解析与执行会阻塞文档的解析，也就是说，在构建 DOM 时，HTML 解析器若遇到了 JavaScript，那么
它会暂停文档的解析，将控制权移交给 JavaScript 引擎，等 JavaScript 引擎运行完毕，浏览器再从中断的地方恢复继续解
析文档。

也就是说，如果你想首屏渲染的越快，就越不应该在首屏就加载 JS 文件，这也是都建议将 script 标签放在 body 标签底部的
原因。当然在当下，并不是说 script 标签必须放在底部，因为你可以给 script 标签添加 defer 或者 async 属性。

19. async 和 defer 的作用是什么？有什么区别？（浏览器解析过程）

（1）js脚本没有 defer 或 async，浏览器会立即加载并执行指定的脚本，也就是说不等待后续载入的文档元素，读到就加载并执
行。

（2）defer 属性表示延迟执行引入的 JavaScript，即这段 JavaScript 加载时 HTML 并未停止解析，这两个过程是并行的。
当整个 document 解析完毕后再执行脚本文件，在 DOMContentLoaded 事件触发之前完成。多个脚本按顺序执行。

（3）async 属性表示异步执行引入的 JavaScript，与 defer 的区别在于，如果已经加载好，就会开始执行，也就是说它的执
行仍然会阻塞文档的解析，只是它的加载过程不会阻塞。多个脚本的执行顺序无法保证。

详细资料可以参考：
《defer 和 async 的区别》

20. 什么是文档的预解析？（浏览器解析过程）

Webkit 和 Firefox 都做了这个优化，当执行 JavaScript 脚本时，另一个线程解析剩下的文档，并加载后面需要通过网络加
载的资源。这种方式可以使资源并行加载从而使整体速度更快。需要注意的是，预解析并不改变 DOM 树，它将这个工作留给主解析
过程，自己只解析外部资源的引用，比如外部脚本、样式表及图片。

21. CSS 如何阻塞文档解析？（浏览器解析过程）

理论上，既然样式表不改变 DOM 树，也就没有必要停下文档的解析等待它们，然而，存在一个问题，JavaScript 脚本执行时可
能在文档的解析过程中请求样式信息，如果样式还没有加载和解析，脚本将得到错误的值，显然这将会导致很多问题。

所以如果浏览器尚未完成 CSSOM 的下载和构建，而我们却想在此时运行脚本，那么浏览器将延迟 JavaScript 脚本执行和文档
的解析，直至其完成 CSSOM 的下载和构建。也就是说，在这种情况下，浏览器会先下载和构建 CSSOM，然后再执行 JavaScript，
最后再继续文档的解析。

22. 渲染页面时常见哪些不良现象？（浏览器渲染过程）

FOUC：主要指的是样式闪烁的问题，由于浏览器渲染机制（比如firefox），在 CSS 加载之前，先呈现了 HTML，就会导致展示
出无样式内容，然后样式突然呈现的现象。会出现这个问题的原因主要是 css 加载时间过长，或者 css 被放在了文档底
部。

白屏：有些浏览器渲染机制（比如chrome）要先构建 DOM 树和 CSSOM 树，构建完成后再进行渲染，如果 CSS 部分放在 HTML
尾部，由于 CSS 未加载完成，浏览器迟迟未渲染，从而导致白屏；也可能是把 js 文件放在头部，脚本的加载会阻塞后面
文档内容的解析，从而页面迟迟未渲染出来，出现白屏问题。

详细资料可以参考：
《前端魔法堂：解秘 FOUC》
《白屏问题和 FOUC》

23. 如何优化关键渲染路径？（浏览器渲染过程）

为尽快完成首次渲染，我们需要最大限度减小以下三种可变因素：

（1）关键资源的数量。
（2）关键路径长度。
（3）关键字节的数量。

关键资源是可能阻止网页首次渲染的资源。这些资源越少，浏览器的工作量就越小，对 CPU 以及其他资源的占用也就越少。

同样，关键路径长度受所有关键资源与其字节大小之间依赖关系图的影响：某些资源只能在上一资源处理完毕之后才能开始下载，
并且资源越大，下载所需的往返次数就越多。

最后，浏览器需要下载的关键字节越少，处理内容并让其出现在屏幕上的速度就越快。要减少字节数，我们可以减少资源数（将它
们删除或设为非关键资源），此外还要压缩和优化各项资源，确保最大限度减小传送大小。

优化关键渲染路径的常规步骤如下：

（1）对关键路径进行分析和特性描述：资源数、字节数、长度。
（2）最大限度减少关键资源的数量：删除它们，延迟它们的下载，将它们标记为异步等。
（3）优化关键字节数以缩短下载时间（往返次数）。
（4）优化其余关键资源的加载顺序：您需要尽早下载所有关键资产，以缩短关键路径长度。

详细资料可以参考：
《优化关键渲染路径》

24. 什么是重绘和重排（回流）？（浏览器绘制过程）

重绘：当渲染树中的一些元素需要更新属性，而这些属性只是影响元素的外观、风格，而不会影响布局的操作，比如 background
-color，我们将这样的操作称为重绘。

重排（回流）：当渲染树中的一部分（或全部）因为元素的规模尺寸、布局、隐藏等改变而需要重新构建的操作，会影响到布局的操作，这样
的操作我们称为回流。

常见引起回流属性和方法：

任何会改变元素几何信息（元素的位置和尺寸大小）的操作，都会触发回流。

（1）添加或者删除可见的 DOM 元素；
（2）元素尺寸改变——边距、填充、边框、宽度和高度
（3）内容变化，比如用户在 input 框中输入文字
（4）浏览器窗口尺寸改变——resize事件发生时
（5）计算 offsetWidth 和 offsetHeight 属性
（6）设置 style 属性的值
（7）当你修改网页的默认字体时。

回流必定会发生重绘，重绘不一定会引发回流。回流所需的成本比重绘高的多，改变父节点里的子节点很可能会导致父节点的一系列
回流。

常见引起重绘属性和方法：

color	border-style	visibility	background
text-decoration	background-image	background-position	background-repeat
outline-color	outline	outline-style	border-radius
outline-width	box-shadow	background-size	

常见引起回流属性和方法：

常见引起重排属性和方法			
width	height	margin	padding
display	border	position	overflow
clientWidth	clientHeight	clientTop	clientLeft
offsetWidth	offsetHeight	offsetTop	offsetLeft
scrollWidth	scrollHeight	scrollTop	scrollLeft
scrollIntoView()	scrollTo()	getComputedStyle()	
getBoundingClientRect()	scrollIntoViewIfNeeded()		

详细资料可以参考：
《浏览器的回流与重绘》

25. 如何减少回流？（浏览器绘制过程）

（1）使用 transform 替代 top

（2）不要把节点的属性值放在一个循环里当成循环里的变量

（3）不要使用 table 布局，可能很小的一个小改动会造成整个 table 的重新布局

（4）把 DOM 离线后修改。如：使用 documentFragment 对象在内存里操作 DOM

（5）不要一条一条地修改 DOM 的样式。与其这样，还不如预先定义好 css 的 class，然后修改 DOM 的 className。

26. 为什么操作 DOM 慢？（浏览器绘制过程）

一些 DOM 的操作或者属性访问可能会引起页面的回流和重绘，从而引起性能上的消耗。

27. DOMContentLoaded 事件和 Load 事件的区别?

当初始的 HTML 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发，而无需等待样式表、图像和
子框架的加载完成。

Load 事件是当所有资源加载完成后触发的。

详细资料可以参考：
《DOMContentLoaded 事件 和 Load 事件的区别？》

28. 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。

    新增的有：

    画布： canvas;
    音视频： video 和 audio 元素;
    web存储：本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失;
            sessionStorage 的数据在浏览器关闭后自动删除;
    语义化更好的内容元素，比如 article、footer、header、main、nav、section;
    表单控件，calendar、date、time、email、url、search;
    新的技术 webworker, websocket;
    新的文档属性 document.visibilityState

    移除的元素有：

    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；
```

29. 如何处理 HTML5 新标签的浏览器兼容问题?

```
    （1） 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。

    （2） 当然也可以直接使用成熟的框架，比如 html5shiv ;
        `<!--[if lt IE 9]>
        <script>
src="https://cdn.jsdelivr.net/npm/html5shiv/dist/html5shiv.min.js"</script>
        <![endif]-->`

        [if lte IE 9]……[endif] 判断 IE 的版本，限定只有 IE9 以下浏览器版本需要执行的语句。
```

30. 简述一下你对 HTML 语义化的理解?

相关知识点：

```
（1） 用正确的标签做正确的事情。
（2） html 语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析;
（3） 即使在没有样式 CSS 情况下也以一种文档格式显示，并且是容易阅读的;
（4） 搜索引擎的爬虫也依赖于 HTML 标记来确定上下文和各个关键字的权重，利于 SEO（搜索引擎优化） ;
（5） 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。
```

回答：

我认为 html 语义化主要指的是我们应该使用合适的标签来划分网页内容的结构。html 的本质作用其实就是定义网页文档的结构，
一个语义化的文档，能够使页面的结构更加清晰，易于理解。这样不仅有利于开发者的维护和理解，同时也能够使机器对文档内容进
行正确的解读。比如说我们常用的 b 标签和 strong 标签，它们在样式上都是文字的加粗，但是 strong 标签拥有强调的语义。
对于一般显示来说，可能我们看上去没有差异，但是对于机器来说，就会有很大的不同。如果用户使用的是屏幕阅读器来访问网页的
话，使用 strong 标签就会有明显的语调上的变化，而 b 标签则没有。如果是搜索引擎的爬虫对我们网页进行分析的话，那么它会
依赖于 html 标签来确定上下文和各个关键字的权重，一个语义化的文档对爬虫来说是友好的，是有利于爬虫对文档内容解读的，
从而有利于我们网站的 SEO 。从 html5 我们可以看出，标准是倾向于以语义化的方式来构建网页的，比如新增了 header 、fo
oter 这些语义标签，删除了 big 、font 这些没有语义的标签。

详细资料可以参考：
《语义化的 HTML 结构到底有什么好处？》
《如何理解 Web 语义化？》
《我的 HTML 会说话——从实用出发，谈谈 HTML 的语义化》

31. b 与 strong 的区别和 i 与 em 的区别?

从页面显示效果来看，被 <b> 和 包围的文字将会被加粗，而被 <i> 和 包围的文字将以斜体的形式呈现。

但是 <b> <i> 是自然样式标签，分别表示无意义的加粗，无意义的斜体，表现样式为 { font-weight: bolder}，仅仅表示「这
里应该用粗体显示」或者「这里应该用斜体显示」，此两个标签在 HTML4.01 中并不被推荐使用。

而 和 是语义样式标签。 表示一般的强调文本，而 表示比 语义更强的强调文本。

使用阅读设备阅读网页时： 会重读，而 <b> 是展示强调内容。

详细资料可以参考：
《HTML5 中的 b/strong，i/em 有什么区别？》

32. 前端需要注意哪些 SEO ?

（1）合理的 title、description、keywords：搜索对这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。

（2）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。

（3）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，保证重要内容肯定被
抓取。

（4）重要内容不要用 js 输出：爬虫不会执行 js 获取内容

（5）少用 iframe：搜索引擎不会抓取 iframe 中的内容

（6）非装饰性图片必须加 alt

（7）提高网站速度：网站速度是搜索引擎排序的一个重要指标

33. 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：

（1）创建一个和 html 同名的 manifest 文件，然后在页面头部像下面一样加入一个 manifest 的属性。

```
<html lang="en" manifest="index.manifest">
```

（2）在如下 cache.manifest 文件的编写离线存储的资源。

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

CACHE：表示需要离线存储的资源列表，由于包含 manifest 文件的页面将被自动离线存储，所以不需要把页面自身也列出来。

NETWORK：表示在它下面列出来的资源只有在在线的情况下才能访问，他们不会被离线存储，所以在离线情况下无法使用这些资源。不过，如果在 CACHE 和 NETWORK 中有一个相同的资源，那么这个资源还是会被离线存储，也就是说 CACHE 的优先级更高。

FALLBACK：表示如果访问第一个资源失败，那么就使用第二个资源来替换他，比如上面这个文件表示的就是如果访问根目录下任何一个资源失败了，那么就去访问 offline.html 。

（3）在离线状态时，操作 window.applicationCache 进行离线缓存的操作。

如何更新缓存：

（1）更新 manifest 文件
（2）通过 javascript 操作
（3）清除浏览器缓存

注意事项：

（1）浏览器对缓存数据的容量限制可能不太一样（某些浏览器设置的限制是每个站点 5MB）。
（2）如果 manifest 文件，或者内部列举的某一个文件不能正常下载，整个更新过程都将失败，浏览器继续全部使用老的缓存。
（3）引用 manifest 的 html 必须与 manifest 文件同源，在同一个域下。
（4）FALLBACK 中的资源必须和 manifest 文件同源。
（5）当一个资源被缓存后，该浏览器直接请求这个绝对路径也会访问缓存中的资源。
（6）站点中的其他页面即使没有设置 manifest 属性，请求的资源如果在缓存中也从缓存中访问。
（7）当 manifest 文件发生改变时，资源请求本身也会触发更新。

详细的使用可以参考：
《HTML5 离线缓存-manifest 简介》
《有趣的 HTML5：离线存储》

34. 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。

35. 常见的浏览器端的存储技术有哪些?

浏览器常见的存储技术有 cookie、localStorage 和 sessionStorage。

还有两种存储技术用于大规模数据存储，webSQL（已被废除）和 indexDB。

IE 支持 userData 存储数据，但是基本很少使用到，除非有很强的浏览器兼容需求。

详细的资料可以参考：
《很全很全的前端本地存储讲解》

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

相关资料：

SessionStorage， LocalStorage， Cookie 这三者都可以被用来在浏览器端存储数据，而且都是字符串类型的键值对。区别
在于前两者属于 HTML5 WebStorage，创建它们的目的便于客户端存储数据。而 cookie 是网站为了标示用户身份而储存在用户
本地终端上的数据（通常经过加密）。cookie 数据始终在同源（协议、主机、端口相同）的 http 请求中携带（即使不需要），会
在浏览器和服务器间来回传递。

存储大小：
cookie 数据大小不能超过4 k 。

```
        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  数据在页面会话结束时会被清除。页面会话在浏览器打开期间一直保持，并且重新加载或恢复页面仍会
                        保持原来的页面会话。在新标签或窗口打开一个页面时会在顶级浏览上下文中初始化一个新的会话。
        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。

    作用域：
        sessionStorage  只在同源的同窗口（或标签页）中共享数据，也就是只在当前会话中共享。
        localStorage    在所有同源窗口中都是共享的。
        cookie          在所有同源窗口中都是共享的。
```

回答：

```
    浏览器端常用的存储技术是 cookie 、localStorage 和 sessionStorage。

    cookie 其实最开始是服务器端用于记录用户状态的一种方式，由服务器设置，在客户端存储，然后每次发起同源请求时，发送给服
    务器端。cookie 最多能存储 4 k 数据，它的生存时间由 expires 属性指定，并且 cookie 只能被同源的页面访问共享。

    sessionStorage 是 html5 提供的一种浏览器本地存储的方法，它借鉴了服务器端 session 的概念，代表的是一次会话中所保
    存的数据。它一般能够存储 5M 或者更大的数据，它在当前窗口关闭后就失效了，并且 sessionStorage 只能被同一个窗口的同源
    页面所访问共享。

    localStorage 也是 html5 提供的一种浏览器本地存储的方法，它一般也能够存储 5M 或者更大的数据。它和 sessionStorage
    不同的是，除非手动删除它，否则它不会失效，并且 localStorage 也只能被同源页面所访问共享。

    上面几种方式都是存储少量数据的时候的存储方式，当我们需要在本地存储大量数据的时候，我们可以使用浏览器的 indexDB 这是浏
    览器提供的一种本地的数据库存储机制。它不是关系型数据库，它内部采用对象仓库的形式存储数据，它更接近 NoSQL 数据库。
```

详细的资料可以参考：
[《请描述一下 cookies，sessionStorage 和 localStorage 的区别？》]
[《浏览器数据库 IndexedDB 入门教程》]

37. iframe 有那些缺点?

iframe 元素会创建包含另外一个文档的内联框架（即行内框架）。

主要缺点有：

（1） iframe 会阻塞主页面的 onload 事件。window 的 onload 事件需要在所有 iframe 加载完毕后（包含里面的元素）才
会触发。在 Safari 和 Chrome 里，通过 JavaScript 动态设置 iframe 的 src 可以避免这种阻塞情况。
（2） 搜索引擎的检索程序无法解读这种页面，不利于网页的 SEO 。
（3） 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。
（4） 浏览器的后退按钮失效。
（5） 小型的移动设备无法完全显示框架。

详细的资料可以参考：
《使用 iframe 的优缺点》
《iframe 简单探索以及 iframe 跨域处理》

38. Label 的作用是什么？是怎么用的？

label 标签来定义表单控制间的关系，当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。

```
<label for="Name">Number:</label>
<input type="text" name="Name" id="Name"/>
```

39. HTML5 的 form 的自动完成功能是什么？

autocomplete 属性规定输入字段是否应该启用自动完成功能。默认为启用，设置为 autocomplete=off 可以关闭该功能。

自动完成允许浏览器预测对字段的输入。当用户在字段开始键入时，浏览器基于之前键入过的值，应该显示出在字段中填写的选项。

autocomplete 属性适用于 <form>，以及下面的 <input> 类型：text, search, url, telephone, email, password,
datepickers, range 以及 color。

40. 如何实现浏览器内多个标签页之间的通信？

相关资料：

（1）使用 WebSocket，通信的标签页连接同一个服务器，发送消息到服务器后，服务器推送消息给所有连接的客户端。

（2）使用 SharedWorker （只在 chrome 浏览器实现了），两个页面共享同一个线程，通过向线程发送数据和接收数据来实现标
签页之间的双向通行。

（3）可以调用 localStorage、cookies 等本地存储方式，localStorge 另一个浏览上下文里被添加、修改或删除时，它都会触
发一个 storage 事件，我们通过监听 storage 事件，控制它的值来进行页面信息通信；

（4）如果我们能够获得对应标签页的引用，通过 postMessage 方法也是可以实现多个标签页通信的。

回答：

实现多个标签页之间的通信，本质上都是通过中介者模式来实现的。因为标签页之间没有办法直接通信，因此我们可以找一个中介者，
让标签页和中介者进行通信，然后让这个中介者来进行消息的转发。

第一种实现的方式是使用 websocket 协议，因为 websocket 协议可以实现服务器推送，所以服务器就可以用来当做这个中介者。
标签页通过向服务器发送数据，然后由服务器向其他标签页推送转发。

第二种是使用 SharedWorker 的方式，sharedworker 会在页面存在的生命周期内创建一个唯一的线程，并且开启多个页面也只会使
用同一个线程。这个时候共享线程就可以充当中介者的角色。标签页间通过共享一个线程，然后通过这个共享的线程来实现数据的交
换。

第三种方式是使用 localStorage 的方式，我们可以在一个标签页对 localStorage 的变化事件进行监听，然后当另一个标签页
修改数据的时候，我们就可以通过这个监听事件来获取到数据。这个时候 localStorage 对象就是充当的中介者的角色。

还有一种方式是使用 postMessage 方法，如果我们能够获得对应标签页的引用，我们就可以使用 postMessage 方法，进行通信。

详细的资料可以参考：

《WebSocket 教程》
《WebSocket 协议：5分钟从入门到精通》
《WebSocket 学习（一）——基于 socket.io 实现简单多人聊天室》
《使用 Web Storage API》
《JavaScript 的多线程，Worker 和 SharedWorker》
《实现多个标签页之间通信的几种方法》

41. webSocket 如何兼容低版本浏览器?

Adobe Flash Socket 、
ActiveX HTMLFile (IE) 、
基于 multipart 编码发送 XHR 、
基于长轮询的 XHR

42. 页面可见性（Page Visibility API） 可以有哪些用途?

这个新的 API 的意义在于，通过监听网页的可见性，可以预判网页的卸载，还可以用来节省资源，减缓电能的消耗。比如，一旦用户
不看网页，下面这些网页行为都是可以暂停的。

（1）对服务器的轮询
（2）网页动画
（3）正在播放的音频或视频

详细资料可以参考：
《Page Visibility API 教程》

43. 如何在页面上实现一个圆形的可点击区域?

（1）纯 html 实现，使用 <area> 来给 <img> 图像标记热点区域的方式，<map> 标签用来定义一个客户端图像映射，<area>
标签用来定义图像映射中的区域，area 元素永远嵌套在 map 元素内部，我们可以将 area 区域设置为圆形，从而实现可点击
的圆形区域。

（2）纯 css 实现，使用 border-radius ，当 border-radius 的长度等于宽高相等的元素值的一半时，即可实现一个圆形的
点击区域。

（3）纯 js 实现，判断一个点在不在圆上的简单算法，通过监听文档的点击事件，获取每次点击时鼠标的位置，判断该位置是否在我
们规定的圆形区域内。

详细资料可以参考：
《如何在页面上实现一个圆形的可点击区域？》
《HTML 标签及在实际开发中的应用》

44. 实现不使用 border 画出 1 px 高的线，在不同浏览器的标准模式与怪异模式下都能保持一致的效果。

```
<div style="height:1px;overflow:hidden;background:red"></div>
```

45. title 与 h1 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。

46. <img> 的 title 和 alt 有什么区别?

title 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（图片无法加载时显示文字）、读屏器阅读图片。可
提高图片可访问性，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。

47. Canvas 和 SVG 有什么区别?

Canvas 是一种通过 JavaScript 来绘制 2D 图形的方法。Canvas 是逐像素来进行渲染的，因此当我们对 Canvas 进行缩放时，
会出现锯齿或者失真的情况。

SVG 是一种使用 XML 描述 2D 图形的语言。SVG 基于 XML，这意味着 SVG DOM 中的每个元素都是可用的。我们可以为某个元素
附加 JavaScript 事件监听函数。并且 SVG 保存的是图形的绘制方法，因此当 SVG 图形缩放时并不会失真。

详细资料可以参考：
《SVG 与 HTML5 的 canvas 各有什么优点，哪个更有前途？》

48. 网页验证码是干嘛的，是为了解决什么安全问题?

（1）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
（2）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

49. 渐进增强和优雅降级的定义

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：一开始就根据高版本浏览器构建完整的功能，然后再针对低版本浏览器进行兼容。

50. attribute 和 property 的区别是什么?

attribute 是 dom 元素在文档中作为 html 标签拥有的属性；
property 就是 dom 元素在 js 中作为对象拥有的属性。
对于 html 的标准属性来说，attribute 和 property 是同步的，是会自动更新的，
但是对于自定义的属性来说，他们是不同步的。

51. 对 web 标准可用性、可访问性、可维护性的理解

可用性（Usability）：产品是否容易上手，用户能否完成任务，效率如何，以及这过程中用户的主观感受可好，是从用户的角度来看
产品的质量。可用性好意味着产品质量高，是企业的核心竞争力

可访问性（Accessibility）：Web 内容对于残障用户的可阅读和可理解性

可维护性（Maintainability）：一般包含两个层次，一是当系统出现问题时，快速定位并解决问题的成本，成本低则可维护性好。
二是代码是否容易被人理解，是否容易修改和增强功能。

52. IE 各版本和 Chrome 可以并行下载多少个资源?

（1） IE6 2 个并发
（2） iE7 升级之后的 6 个并发，之后版本也是 6 个
（3） Firefox，chrome 也是6个

53. Flash、Ajax 各自的优缺点，在使用中如何取舍?

Flash：
（1） Flash 适合处理多媒体、矢量图形、访问机器
（2） 对 CSS、处理文本上不足，不容易被搜索

Ajax：
（1） Ajax 对 CSS、文本支持很好，支持搜索
（2） 多媒体、矢量图形、机器访问不足

共同点：
（1） 与服务器的无刷新传递消息
（2） 可以检测用户离线和在线状态
（3） 操作 DOM

54. 怎么重构页面?

```
（1） 编写 CSS
（2） 让页面结构更合理化，提升用户体验
（3） 实现良好的页面效果和提升性能
```

55. 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
```

56. 常用的 meta 标签

```
    <meta> 元素可提供有关页面的元信息（meta-information），比如针对搜索引擎和更新频度的描述和关键词。
    <meta> 标签位于文档的头部，不包含任何内容。<meta> 标签的属性定义了与文档相关联的名称/值对。

    <!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=""/>       页面关键词者
    <meta name="author" content="name, email@gmail.com"/>     网页作
    <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 点击无高光
    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">
```

详细资料可以参考：
《Meta 标签用法大全》

57. css reset 和 normalize.css 有什么区别?

相关知识点：

```
    为什么会有 CSS Reset 的存在呢？那是因为早期的浏览器支持和理解的 CSS 规范不同，导致渲染页面时效果不一致，会出现很多
    兼容性问题。

    reset 的目的，是将所有的浏览器的自带样式重置掉，这样更易于保持各浏览器渲染的一致性。

    normalize 的理念则是尽量保留浏览器的默认样式，不进行太多的重置，而尽力让这些样式保持一致并尽可能与现代标准相符合。

    1.Normalize.css 保护了有价值的默认值

    Reset 通过为几乎所有的元素施加默认样式，强行使得元素有相同的视觉效果。 相比之下，Normalize.css 保持了许多默认的浏
    览器样式。 这就意味着你不用再为所有公共的排版元素重新设置样式。 当一个元素在不同的浏览器中有不同的默认值时，Normali
    ze.css 会力求让这些样式保持一致并尽可能与现代标准相符合。

    2.Normalize.css 修复了浏览器的 bug

    它修复了常见的桌面端和移动端浏览器的 bug。这往往超出了 Reset 所能做到的范畴。关于这一点，Normalize.css 修复的问题
    包含了 HTML5 元素的显示设置、预格式化文字的 font-size 问题、在 IE9 中 SVG 的溢出、许多出现在各浏览器和操作系统中
    的与表单相关的 bug。

    3.Normalize.css 没有复杂的继承链

    使用 Reset 最让人困扰的地方莫过于在浏览器调试工具中大段大段的继承链。在 Normalize.css 中就不会有这样的问题，因为在
    我们的准则中对多选择器的使用是非常谨慎的，我们仅会有目的地对目标元素设置样式。

    4.Normalize.css 是模块化的

    这个项目已经被拆分为多个相关却又独立的部分，这使得你能够很容易也很清楚地知道哪些元素被设置了特定的值。因此这能让你自己
    选择性地移除掉某些永远不会用到部分（比如表单的一般化）。

    5.Normalize.css 拥有详细的文档
```

Normalize.css 的代码基于详细而全面的跨浏览器研究与测试。这个文件中拥有详细的代码说明并在 Github Wiki 中有进一步的
说明。这意味着你可以找到每一行代码具体完成了什么工作、为什么要写这句代码、浏览器之间的差异，并且你可以更容易地进行自己
的测试。

回答：

css reset 是最早的一种解决浏览器间样式不兼容问题的方案，它的基本思想是将浏览器的所有样式都重置掉，从而达到所有浏览器
样式保持一致的效果。但是使用这种方法，可能会带来一些性能上的问题，并且对于一些元素的不必要的样式的重置，其实反而会造成
画蛇添足的效果。

后面出现一种更好的解决浏览器间样式不兼容的方法，就是 normalize.css ，它的思想是尽量的保留浏览器自带的样式，通过在原
有的样式的基础上进行调整，来保持各个浏览器间的样式表现一致。相对与 css reset，normalize.css 的方法保留了有价值的默
认值，并且修复了一些浏览器的 bug，而且使用 normalize.css 不会造成元素复杂的继承链。

详细资料可以参考：
《关于CSS Reset 那些事（一）之 历史演变与 Normalize.css》
《Normalize.css 和 Reset CSS 有什么本质区别没？》

58. 用于预格式化文本的标签是?

预格式化就是保留文字在源码中的格式 最后显示出来样式与源码中的样式一致 所见即所得。

<pre> 定义预格式文本，保持文本原有的格式

59. DHTML 是什么?

DHTML 将 HTML、JavaScript、DOM 以及 CSS 组合在一起，用于创造动态性更强的网页。通过 JavaScript 和 HTML DOM，能
够动态地改变 HTML 元素的样式。

DHTML 实现了网页从 Web 服务器下载后无需再经过服务器的处理，而在浏览器中直接动态地更新网页的内容、排版样式和动画的功
能。例如，当鼠标指针移到文章段落中时，段落能够变成蓝色，或者当鼠标指针移到一个超级链接上时，会自动生成一个下拉式子链
接目录等。

包括：
（1）动态内容（Dynamic Content）：动态地更新网页内容，可“动态”地插入、修改或删除网页的元件，如文字、图像、标记等。

（2）动态排版样式（Dynamic Style Sheets）：W3C 的 CSS 样式表提供了设定 HTML 标记的字体大小、字形、样式、粗细、
文字颜色、行高度、加底线或加中间横线、缩排、与边缘距离、靠左右或置中、背景图片或颜色等排版功能，而“动态排版样
式”即可以“动态”地改变排版样式。

60. head 标签中必不少的是?

<head> 标签用于定义文档的头部，它是所有头部元素的容器。<head> 中的元素可以引用脚本、指示浏览器在哪里找到样式表、提供
元信息等等。

文档的头部描述了文档的各种属性和信息，包括文档的标题、在 Web 中的位置以及和其他文档的关系等。绝大多数文档头部包含的数
据都不会真正作为内容显示给读者。

下面这些标签可用在 head 部分：<base>, <link>, <meta>, <script>, <style>, 以及 <title>。

<title> 定义文档的标题，它是 head 部分中唯一必需的元素。

61. HTML5 新增的表单元素有?

datalist 规定输入域的选项列表，通过 option 创建!

keygen 提供一种验证用户的可靠方法，密钥对生成器，私钥存于客户端，公钥发到服务器，用于之后验证客户端证书!

output 元素用于不同类型的输出!

62. 在 HTML5 中，哪个方法用于获得用户的当前位置?

```
getCurrentPosition()
```

63. 文档的不同注释方式?

```
HTML 的注释方法 <!--注释内容-->

CSS 的注释方法 /*注释内容*/

JavaScript 的注释方法 /* 多行注释方式 */ //单行注释方式
```

64. disabled 和 readonly 的区别?

disabled 指当 input 元素加载时禁用此元素。input 内容不会随着表单提交。

readonly 规定输入字段为只读。input 内容会随着表单提交。

无论设置 readonly 还是 disabled，通过 js 脚本都能更改 input 的 value

65. 主流浏览器内核私有属性 css 前缀?

```
mozilla 内核 （firefox,flock 等）     -moz
webkit  内核 （safari,chrome 等）    -webkit
opera   内核 （opera 浏览器）         -o
trident 内核 （ie 浏览器）            -ms
```

66. 前端性能优化?

前端性能优化主要是为了提高页面的加载速度，优化用户的访问体验。我认为可以从这些方面来进行优化。

第一个方面是页面的内容方面

（1）通过文件合并、css 雪碧图、使用 base64 等方式来减少 HTTP 请求数，避免过多的请求造成等待的情况。

（2）通过 DNS 缓存等机制来减少 DNS 的查询次数。

（3）通过设置缓存策略，对常用不变的资源进行缓存。

（4）使用延迟加载的方式，来减少页面首屏加载时需要请求的资源。延迟加载的资源当用户需要访问时，再去请求加载。

（5）通过用户行为，对某些资源使用预加载的方式，来提高用户需要访问资源时的响应速度。

第二个方面是服务器方面

（1）使用 CDN 服务，来提高用户对于资源请求时的响应速度。

（2）服务器端启用 Gzip、Deflate 等方式对于传输的资源进行压缩，减小文件的体积。

（3）尽可能减小 cookie 的大小，并且通过将静态资源分配到其他域名下，来避免对静态资源请求时携带不必要的 cookie

第三个方面是 CSS 和 JavaScript 方面

（1）把样式表放在页面的 head 标签中，减少页面的首次渲染的时间。

（2）避免使用 @import 标签。

（3）尽量把 js 脚本放在页面底部或者使用 defer 或 async 属性，避免脚本的加载和执行阻塞页面的渲染。

（4）通过对 JavaScript 和 CSS 的文件进行压缩，来减小文件的体积。

详细的资料可以参考：
《前端性能优化之雅虎35条军规》
《你真的了解 gzip 吗？》
《前端性能优化之 gzip》

67. Chrome 中的 Waterfall ?

详细资料可以参考：
《前端性能之 Chrome 的 Waterfall》
《教你读懂网络请求的瀑布图》 《前端妹子跟我抱怨她们的页面加载很慢的时候，如何在她面前优雅地装逼？》

68. 扫描二维码登录网页是什么原理，前后两个事件是如何联系的?

核心过程应该是：浏览器获得一个临时 id，通过长连接等待客户端扫描带有此 id 的二维码后，从长连接中获得客户端上报给 server的帐号信息进行展示。并在客户端点击确认后，获得服务器授信的令牌，进行随后的信息交互过程。在超时、网络断开、其他设备上登录后，此前获得的令牌或丢失、或失效，对授权过程形成有效的安全防护。

我的理解

二维码登录网页的基本原理是，用户进入登录网页后，服务器生成一个 uid 来标识一个用户。对应的二维码对应了一个对应 uid的链接，任何能够识别二维码的应用都可以获得这个链接，但是它们没有办法和对应登录的服务器响应。比如微信的二维码登录，只有用微信识这个二维码才有效。当微信客户端打开这个链接时，对应的登录服务器就获得了用户的相关信息。这个时候登录网页根据先前的长连接获取到服务器传过来的用户信息进行显示。然后提前预加载一些登录后可能用到的信息。当客户端点击确认授权登陆后，服务器生成一个权限令牌给网页，网页之后使用这个令牌进行信息的交互过程。由于整个授权的过程都是在手机端进行的，因此能够很好的防止 PC 上泛滥的病毒。并且在超时、网络断开、其他设备上登录后，此前获得的令牌或丢失、或失效，对授权过程能够形成有效的安全防护。

详细资料可以参考：
《微信扫描二维码登录网页》

69. Html 规范中为什么要求引用资源不加协议头`http`或者`https`?

如果用户当前访问的页面是通过 HTTPS 协议来浏览的，那么网页中的资源也只能通过 HTTPS 协议来引用，否则浏览器会出现警告信息，不同浏览器警告信息展现形式不同。

为了解决这个问题，我们可以省略 URL 的协议声明，省略后浏览器照样可以正常引用相应的资源，这项解决方案称为 protocol-relative URL，暂且可译作协议相对 URL。

如果使用协议相对 URL，无论是使用 HTTPS，还是 HTTP 访问页面，浏览器都会以相同的协议请求页面中的资源，避免弹出类似的警告信息，同时还可以节省5字节的数据量。

详细资料可以参考：
《协议相对 URL》
《Why you need protocol-relative URLs *now*》