目录

1.HTML 面试知识点总结

1.1 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。

<!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。

1.2 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。

1.3 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。

1.4 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。

1.5 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）。

1.6 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

1.7 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

1.8 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：行内元素设置 width 无效，height 无效（可以设置 line-height），设置 margin 和 padding 的上下不会对其他元素产生影响。

1.9 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

1.10 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

1.11 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。

1.12 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。

1.13 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。

1.14. 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。

1.15 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 Trident，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 Blink 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》

1.16 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。

1.17 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》

1.18 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。

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

（1）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》

1.20 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。

1.21 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。

1.22 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》

1.23 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》

1.24 什么是重绘和回流？（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。

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：

常见引起回流属性和方法：

详细资料可以参考：《浏览器的回流与重绘》

1.25 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。

1.26 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。

1.27 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？》

1.28 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：

绘画 canvas;

用于媒介回放的 video 和 audio 元素;

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;

语意化更好的内容元素，比如 article、footer、header、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；

1.29 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 html5shim `<!--[if lt IE 9]> <script> src="http://html5shim.googlecode.com/svn/trunk/html5.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。

1.30 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 ;

（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 、footer 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》

1.31 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？》

1.32 前端需要注意哪些 SEO ？

（1）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。

（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

1.33 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》

1.34 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。

1.35. 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》

1.36 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》

1.37 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》

1.38 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"/>
```

1.39 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。

1.40 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 ShareWorker 的方式，shareWorker 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》

1.41. 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
```

1.42. 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》

1.43. 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》

1.44. 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>
```

1.45. 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。

1.46. <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、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。

1.47. 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？》

1.48. 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

1.49 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。

1.50. 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。

1.51. 对 web 标准、可用性、可访问性的理解

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。

1.52. 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

1.53. 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

1.54. 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

1.55. 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

1.56. 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》

1.57. 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，Normalize.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时非常谨慎的，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？》

1.58. 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

1.59. 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。

1.60. 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。

1.61 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！

1.62. 在 HTML5 中，哪个方法用于获得用户的当前位置？

```
getCurrentPosition()
```

1.63. 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

1.64. 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

1.65. 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

1.66. 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》

1.67. 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？》

1.68. 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》

1.69. 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》

2. CSS 面试知识点总结

2.1.介绍一下标准的 CSS 的盒子模型？低版本 IE 的盒子模型有什么不同的？

相关知识点：

（1）有两种盒子模型：IE 盒模型（border-box）、W3C 标准盒模型（content-box）

（2）盒模型：分为内容（content）、填充（padding）、边界（margin）、边框（border）四个部分

IE 盒模型和 W3C 标准盒模型的区别：

（1）W3C 标准盒模型：属性 width，height 只包含内容 content，不包含 border 和 padding

（2）IE 盒模型：属性 width，height 包含 content、border 和 padding，指的是 content +padding+border。

在 ie8+浏览器中使用哪个盒模型可以由 box-sizing（CSS 新增的属性）控制，默认值为 content-box，即标准盒模型；
如果将 box-sizing 设为 border-box 则用的是 IE 盒模型。如果在 ie6，7，8 中 DOCTYPE 缺失会将盒子模型解释为 IE
盒子模型。若在页面中声明了 DOCTYPE 类型，所有的浏览器都会把盒模型解释为 W3C 盒模型。
回答：
盒模型都是由四个部分组成的，分别是 margin、border、padding 和 content。

标准盒模型和 IE 盒模型的区别在于设置 width 和 height 时，所对应的范围不同。标准盒模型的 width 和 height 属性的
范围只包含了 content，而 IE 盒模型的 width 和 height 属性的范围包含了 border、padding 和 content。

一般来说，我们可以通过修改元素的 box-sizing 属性来改变元素的盒模型。
详细的资料可以参考：《CSS 盒模型详解》

2.2.CSS 选择符有哪些？

（1）id 选择器（#myid）
（2）类选择器（.myclassname）
（3）标签选择器（div,h1,p）
（4）后代选择器（h1 p）
（5）相邻后代选择器（子）选择器（ul>li）
（6）兄弟选择器（li~a）
（7）相邻兄弟选择器（li+a）
（8）属性选择器（a[rel="external"]）
（9）伪类选择器（a:hover,li:nth-child）
（10）伪元素选择器（::before、::after）
（11）通配符选择器（*）

2.3.::before 和:after 中双冒号和单冒号有什么区别？解释一下这 2 个伪元素的作用。

相关知识点：
单冒号（:）用于 CSS3 伪类，双冒号（::）用于 CSS3 伪元素。（伪元素由双冒号和伪元素名称组成）
双冒号是在当前规范中引入的，用于区分伪类和伪元素。不过浏览器需要同时支持旧的已经存在的伪元素写法，
比如:first-line、:first-letter、:before、:after 等，
而新的在 CSS3 中引入的伪元素则不允许再支持旧的单冒号的写法。

想让插入的内容出现在其它内容前，使用::before，否者，使用::after；
在代码顺序上，::after 生成的内容也比::before 生成的内容靠后。
如果按堆栈视角，::after 生成的内容会在::before 生成的内容之上。
回答：
在 css3 中使用单冒号来表示伪类，用双冒号来表示伪元素。但是为了兼容已有的伪元素的写法，在一些浏览器中也可以使用单冒号
来表示伪元素。

伪类一般匹配的是元素的一些特殊状态，如 hover、link 等，而伪元素一般匹配的特殊的位置，比如 after、before 等。

2.4.伪类与伪元素的区别

css 引入伪类和伪元素概念是为了格式化文档树以外的信息。也就是说，伪类和伪元素是用来修饰不在文档树中的部分，比如，一句
话中的第一个字母，或者是列表中的第一个元素。

伪类用于当已有的元素处于某个状态时，为其添加对应的样式，这个状态是根据用户行为而动态变化的。比如说，当用户悬停在指定的
元素时，我们可以通过:hover 来描述这个元素的状态。

伪元素用于创建一些不在文档树中的元素，并为其添加样式。它们允许我们为元素的某些部分设置样式。比如说，我们可以通过::be
fore 来在一个元素前增加一些文本，并为这些文本添加样式。虽然用户可以看到这些文本，但是这些文本实际上不在文档树中。

有时你会发现伪元素使用了两个冒号（::）而不是一个冒号（:）。这是 CSS3 的一部分，并尝试区分伪类和伪元素。大多数浏览
器都支持这两个值。按照规则应该使用（::）而不是（:），从而区分伪类和伪元素。但是，由于在旧版本的 W3C 规范并未对此进行
特别区分，因此目前绝大多数的浏览器都支持使用这两种方式表示伪元素。
详细资料可以参考：《总结伪类与伪元素》

2.5.CSS 中哪些属性可以继承？

相关资料：
每个 CSS 属性定义的概述都指出了这个属性是默认继承的，还是默认不继承的。这决定了当你没有为元素的属性指定值时该如何计算
值。

当元素的一个继承属性没有指定值时，则取父元素的同属性的计算值。只有文档根元素取该属性的概述中给定的初始值（这里的意思应
该是在该属性本身的定义中的默认值）。

当元素的一个非继承属性（在 Mozillacode 里有时称之为 resetproperty）没有指定值时，则取属性的初始值 initialv
alue（该值在该属性的概述里被指定）。

有继承性的属性：

（1）字体系列属性
font、font-family、font-weight、font-size、font-style、font-variant、font-stretch、font-size-adjust

（2）文本系列属性
text-indent、text-align、text-shadow、line-height、word-spacing、letter-spacing、
text-transform、direction、color

（3）表格布局属性
caption-sideborder-collapseempty-cells

（4）列表属性
list-style-type、list-style-image、list-style-position、list-style

（5）光标属性
cursor

（6）元素可见性
visibility

（7）还有一些不常用的；speak，page，设置嵌套引用的引号类型 quotes 等属性

注意：当一个属性不是继承属性时，可以使用 inherit 关键字指定一个属性应从父元素继承它的值，inherit 关键字用于显式地
指定继承性，可用于任何继承性/非继承性属性。
回答：
每一个属性在定义中都给出了这个属性是否具有继承性，一个具有继承性的属性会在没有指定值的时候，会使用父元素的同属性的值
来作为自己的值。

一般具有继承性的属性有，字体相关的属性，font-size 和 font-weight 等。文本相关的属性，color 和 text-align 等。
表格的一些布局属性、列表属性如 list-style 等。还有光标属性 cursor、元素可见性 visibility。

当一个属性不是继承属性的时候，我们也可以通过将它的值设置为 inherit 来使它从父元素那获取同名的属性值来继承。
详细的资料可以参考： 《继承属性》 《CSS 有哪些属性可以继承？》

2.6.CSS 优先级算法如何计算？

相关知识点：
CSS 的优先级是根据样式声明的特殊性值来判断的。

选择器的特殊性值分为四个等级，如下：

（1）标签内选择符 x,0,0,0
（2）ID 选择符 0,x,0,0
（3）class 选择符/属性选择符/伪类选择符　　0,0,x,0
（4）元素和伪元素选择符 0,0,0,x

计算方法：

（1）每个等级的初始值为 0
（2）每个等级的叠加为选择器出现的次数相加
（3）不可进位，比如 0,99,99,99
（4）依次表示为：0,0,0,0
（5）每个等级计数之间没关联
（6）等级判断从左向右，如果某一位数值相同，则判断下一位数值
（7）如果两个优先级相同，则最后出现的优先级高，!important 也适用
（8）通配符选择器的特殊性值为：0,0,0,0
（9）继承样式优先级最低，通配符样式优先级高于继承样式
（10）!important（权重），它没有特殊性值，但它的优先级是最高的，为了方便记忆，可以认为它的特殊性值为 1,0,0,0,0。

计算实例：

（1）#demoa{color:orange;}/*特殊性值：0,1,0,1*/
（2）div#demoa{color:red;}/*特殊性值：0,1,0,2*/

注意：
（1）样式应用时，css 会先查看规则的权重（!important），加了权重的优先级最高，当权重相同的时候，会比较规则的特殊性。

（2）特殊性值越大的声明优先级越高。

（3）相同特殊性值的声明，根据样式引入的顺序，后声明的规则优先级高（距离元素出现最近的）

(4) 部分浏览器由于字节溢出问题出现的进位表现不做考虑
回答：

判断优先级时，首先我们会判断一条属性声明是否有权重，也就是是否在声明后面加上了!important。一条声明如果加上了权重，
那么它的优先级就是最高的，前提是它之后不再出现相同权重的声明。如果权重相同，我们则需要去比较匹配规则的特殊性。

一条匹配规则一般由多个选择器组成，一条规则的特殊性由组成它的选择器的特殊性累加而成。选择器的特殊性可以分为四个等级，
第一个等级是行内样式，为 1000，第二个等级是 id 选择器，为 0100，第三个等级是类选择器、伪类选择器和属性选择器，为 0010，
第四个等级是元素选择器和伪元素选择器，为 0001。规则中每出现一个选择器，就将它的特殊性进行叠加，这个叠加只限于对应的等
级的叠加，不会产生进位。选择器特殊性值的比较是从左向右排序的，也就是说以 1 开头的特殊性值比所有以 0 开头的特殊性值要大。
比如说特殊性值为 1000 的的规则优先级就要比特殊性值为 0999 的规则高。如果两个规则的特殊性值相等的时候，那么就会根据它们引
入的顺序，后出现的规则的优先级最高。
对于组合声明的特殊性值计算可以参考：《CSS 优先级计算及应用》《CSS 优先级计算规则》《有趣：256 个 class 选择器可以干掉 1 个 id 选择器》

2.7.关于伪类 LVHA 的解释?

a 标签有四种状态：链接访问前、链接访问后、鼠标滑过、激活，分别对应四种伪类:link、:visited、:hover、:active；

当链接未访问过时：

（1）当鼠标滑过 a 链接时，满足:link 和:hover 两种状态，要改变 a 标签的颜色，就必须将:hover 伪类在:link 伪
类后面声明；
（2）当鼠标点击激活 a 链接时，同时满足:link、:hover、:active 三种状态，要显示 a 标签激活时的样式（:active），
必须将:active 声明放到:link 和:hover 之后。因此得出 LVHA 这个顺序。

当链接访问过时，情况基本同上，只不过需要将:link 换成:visited。

这个顺序能不能变？可以，但也只有:link 和:visited 可以交换位置，因为一个链接要么访问过要么没访问过，不可能同时满足，
也就不存在覆盖的问题。

2.8.CSS3 新增伪类有那些？

（1）elem:nth-child(n)选中父元素下的第 n 个子元素，并且这个子元素的标签名为 elem，n 可以接受具体的数
值，也可以接受函数。

（2）elem:nth-last-child(n)作用同上，不过是从后开始查找。

（3）elem:last-child 选中最后一个子元素。

（4）elem:only-child 如果 elem 是父元素下唯一的子元素，则选中之。

（5）elem:nth-of-type(n)选中父元素下第 n 个 elem 类型元素，n 可以接受具体的数值，也可以接受函数。

（6）elem:first-of-type 选中父元素下第一个 elem 类型元素。

（7）elem:last-of-type 选中父元素下最后一个 elem 类型元素。

（8）elem:only-of-type 如果父元素下的子元素只有一个 elem 类型元素，则选中该元素。

（9）elem:empty 选中不包含子元素和内容的 elem 类型元素。

（10）elem:target 选择当前活动的 elem 元素。

（11）:not(elem)选择非 elem 元素的每个元素。

（12）:enabled　控制表单控件的禁用状态。

（13）:disabled　控制表单控件的禁用状态。

(14):checked 单选框或复选框被选中。

详细的资料可以参考：《CSS3 新特性总结(伪类)》《浅谈 CSS 伪类和伪元素及 CSS3 新增伪类》

2.9.如何居中 div？

-水平居中：给 div 设置一个宽度，然后添加 margin:0auto 属性

```
div {
    width: 200px;
    margin: 0auto;
}
```

-水平居中，利用 text-align:center 实现

```
.container {
    background: rgba(0, 0, 0, 0.5);
    text-align: center;
    font-size: 0;
}
```

```
.box {
    display: inline-block;
    width: 500px;
    height: 400px;
    background-color: pink;
}
-让绝对定位的 div 居中
div {
    position: absolute;
    width: 300px;
    height: 300px;
    margin: auto;
    top: 0;
    left: 0;
    bottom: 0;
    right: 0;
    background-color: pink; /*方便看效果*/
}
-水平垂直居中一
/*确定容器的宽高宽 500 高 300 的层设置层的外边距 div{*/position:absolute;/*绝对定位
*/width:500px;height:300px;top:50%;left:50%;margin:-150px00-250px;/*外边距为自身宽高的
一半*/background-color:pink;/*方便看效果*/
}
-水平垂直居中二
/*未知容器的宽高，利用`transform`属性*/div {
    position: absolute; /*相对定位或绝对定位均可*/
    width: 500px;
    height: 300px;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: pink; /*方便看效果*/
}
-水平垂直居中三
/*利用 flex 布局实际使用时应考虑兼容性*/
.container {
    display: flex;
    align-items: center; /*垂直居中*/
    justify-content: center; /*水平居中*/
}
.containerdiv {
    width: 100px;
    height: 100px;
```

```
    background-color: pink; /*方便看效果*/
}
```
-水平垂直居中四
```
/*利用 text-align:center 和 vertical-align:middle 属性*/
.container {
    position: fixed;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    background: rgba(0, 0, 0, 0.5);
    text-align: center;
    font-size: 0;
    white-space: nowrap;
    overflow: auto;
}

.container::after {
    content: "";
    display: inline-block;
    height: 100%;
    vertical-align: middle;
}

.box {
    display: inline-block;
    width: 500px;
    height: 400px;
    background-color: pink;
    white-space: normal;
    vertical-align: middle;
}
```
回答：
一般常见的几种居中的方法有：

对于宽高固定的元素

（1）我们可以利用 margin:0auto 来实现元素的水平居中。

（2）利用绝对定位，设置四个方向的值都为 0，并将 margin 设置为 auto，由于宽高固定，因此对应方向实现平分，可以实现水
平和垂直方向上的居中。

（3）利用绝对定位，先将元素的左上角通过 top:50%和 left:50%定位到页面的中心，然后再

通过 margin 负值来调整元素
的中心点到页面的中心。

（4）利用绝对定位，先将元素的左上角通过 top:50%和 left:50%定位到页面的中心，然后再通过 translate 来调整元素
的中心点到页面的中心。

（5）使用 flex 布局，通过 align-items:center 和 justify-content:center 设置容器的垂直和水平方向上为居中对
齐，然后它的子元素也可以实现垂直和水平的居中。

对于宽高不定的元素，上面的后面两种方法，可以实现元素的垂直和水平的居中。

2.10.display 有哪些值？说明他们的作用。

block 块类型。默认宽度为父元素宽度，可设置宽高，换行显示。
none 元素不显示，并从文档流中移除。
inline 行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示。
inline-block 默认宽度为内容宽度，可以设置宽高，同行显示。
list-item 像块类型元素一样显示，并添加样式列表标记。
table 此元素会作为块级表格来显示。
inherit 规定应该从父元素继承 display 属性的值。
详细资料可以参考： 《CSSdisplay 属性》

2.11.position 的值 relative 和 absolute 定位原点是？

相关知识点：
absolute
生成绝对定位的元素，相对于值不为 static 的第一个父元素的 paddingbox 进行定位，也可以理解为离自己这一级元素最近的
一级 position 设置为 absolute 或者 relative 的父元素的 paddingbox 的左上角为原点的。

fixed（老 IE 不支持）
生成绝对定位的元素，相对于浏览器窗口进行定位。

relative
生成相对定位的元素，相对于其元素本身所在正常位置进行定位。

static
默认值。没有定位，元素出现在正常的流中（忽略 top,bottom,left,right,z-index 声明）。

inherit
规定从父元素继承 position 属性的值。
回答：

relative 定位的元素，是相对于元素本身的正常位置来进行定位的。

absolute 定位的元素，是相对于它的第一个 position 值不为 static 的祖先元素的 paddingbox 来进行定位的。这句话
我们可以这样来理解，我们首先需要找到绝对定位元素的一个 position 的值不为 static 的祖先元素，然后相对于这个祖先元
素的 paddingbox 来定位，也就是说在计算定位距离的时候，padding 的值也要算进去。

2.12.CSS3 有哪些新特性？（根据项目回答）

新增各种 CSS 选择器 （:not(.input)：所有 class 不是“input”的节点）
圆角 （border-radius:8px）
多列布局 （multi-columnlayout）
阴影和反射 （Shadow\Reflect）
文字特效 （text-shadow）
文字渲染 （Text-decoration）
线性渐变 （gradient）
旋转 （transform）
缩放，定位，倾斜，动画，多背景
例如：transform:\scale(0.85,0.90)\translate(0px,-30px)\skew(-9deg,0deg)\Animation:

2.13.请解释一下 CSS3 的 Flexbox（弹性盒布局模型），以及适用场景？

相关知识点：
Flex 是 FlexibleBox 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

任何一个容器都可以指定为 Flex 布局。行内元素也可以使用 Flex 布局。注意，设为 Flex 布局以后，子元素的 float、cl
ear 和 vertical-align 属性将失效。

采用 Flex 布局的元素，称为 Flex 容器（flexcontainer），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex
项目（flexitem），简称"项目"。

容器默认存在两根轴：水平的主轴（mainaxis）和垂直的交叉轴（crossaxis），项目默认沿主轴排列。

以下 6 个属性设置在容器上。

flex-direction 属性决定主轴的方向（即项目的排列方向）。

flex-wrap 属性定义，如果一条轴线排不下，如何换行。

flex-flow 属性是 flex-direction 属性和 flex-wrap 属性的简写形式，默认值为 rownowrap。

justify-content 属性定义了项目在主轴上的对齐方式。

align-items 属性定义项目在交叉轴上如何对齐。

align-content 属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

以下 6 个属性设置在项目上。

order 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 0。

flex-grow 属性定义项目的放大比例，默认为 0，即如果存在剩余空间，也不放大。

flex-shrink 属性定义了项目的缩小比例，默认为 1，即如果空间不足，该项目将缩小。

flex-basis 属性定义了在分配多余空间之前，项目占据的主轴空间。浏览器根据这个属性，计算主轴是否有多余空间。它的默认
值为 auto，即项目的本来大小。

flex 属性是 flex-grow，flex-shrink 和 flex-basis 的简写，默认值为 01auto。

align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父
元素的 align-items 属性，如果没有父元素，则等同于 stretch。
回答：
flex 布局是 CSS3 新增的一种布局方式,我们可以通过将一个元素的 display 属性值设置为 flex 从而使它成为一个 flex
容器，它的所有子元素都会成为它的项目。

一个容器默认有两条轴，一个是水平的主轴，一个是与主轴垂直的交叉轴。我们可以使用 flex-direction 来指定主轴的方向。
我们可以使用 justify-content 来指定元素在主轴上的排列方式，使用 align-items 来指定元素在交叉轴上的排列方式。还
可以使用 flex-wrap 来规定当一行排列不下时的换行方式。

对于容器中的项目，我们可以使用 order 属性来指定项目的排列顺序，还可以使用 flex-grow 来指定当排列空间有剩余的时候，
项目的放大比例。还可以使用 flex-shrink 来指定当排列空间不足时，项目的缩小比例。
详细资料可以参考： 《Flex 布局教程：语法篇》 《Flex 布局教程：实例篇》

2.14.用纯 CSS 创建一个三角形的原理是什么？

采用的是相邻边框连接处的均分原理。

将元素的宽高设为 0，只设置 border，把任意三条边隐藏掉（颜色设为 transparent），剩下的就是一个三角形。

```
#demo {
width: 0;
height: 0;
border-width: 20px;
border-style: solid;
border-color: transparenttransparentredtransparent;
}
```

2.15.一个满屏品字布局如何设计?

简单的方式：

上面的 div 宽 100%，

下面的两个 div 分别宽 50%，

然后用 float 或者 inline 使其不换行即可

2.16.CSS 多列等高如何实现？

（1）利用 padding-bottom|margin-bottom 正负值相抵，不会影响页面布局的特点。设置父容器设置超出隐藏（overflow: hidden），这样父容器的高度就还是它里面的列没有设定 padding-bottom 时的高度，当它里面的任一列高度增加了，则父容器的高度被撑到里面最高那列的高度，其他比这列矮的列会用它们的 padding-bottom 补偿这部分高度差。

（2）利用 table-cell 所有单元格高度都相等的特性，来实现多列等高。

（3）利用 flex 布局中项目 align-items 属性默认为 stretch，如果项目未设置高度或设为 auto，将占满整个容器的高度的特性，来实现多列等高。

详细资料可以参考：《前端应该掌握的 CSS 实现多列等高布局》《CSS：多列等高布局》

2.17.经常遇到的浏览器的兼容性有哪些？原因，解决方法是什么，常用 hack 的技巧？

（1）png24 位的图片在 iE6 浏览器上出现背景

解决方案：做成 PNG8，也可以引用一段脚本处理。

（2）浏览器默认的 margin 和 padding 不同

解决方案：加一个全局的*{margin:0;padding:0;}来统一。

（3）IE6 双边距 bug：在 IE6 下，如果对元素设置了浮动，同时又设置了 margin-left 或 margin-right，margin 值会加倍。

#box{float:left;width:10px;margin:00010px;}

这种情况之下 IE 会产生 20px 的距离
解决方案：在 float 的标签样式控制中加入_display:inline;将其转化为行内属性。(_这个符号只有 ie6 会识别)

（4）渐进识别的方式，从总体中逐渐排除局部。
首先，巧妙的使用"\9"这一标记，将 IE 游览器从所有情况中分离出来。
接着，再次使用"+"将 IE8 和 IE7、IE6 分离开来，这样 IE8 已经独立识别。
.bb{
background-color:#f1ee18;/*所有识别*/
.background-color:#00deff\9;/*IE6、7、8 识别*/
+background-color:#a200ff;/*IE6、7 识别*/
_background-color:#1e0bd1;/*IE6 识别*/
}

（5）IE 下，可以使用获取常规属性的方法来获取自定义属性，也可以使用 getAttribute()获取自定义
属性；Firefox 下，只能使用 getAttribute()获取自定义属性
解决方法：统一通过 getAttribute()获取自定义属性。

（6）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。
解决方法：（条件注释）缺点是在 IE 浏览器下可能会增加额外的 HTTP 请求数。

（7）Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示
解决方法：

1.可通过加入 CSS 属性-webkit-text-size-adjust:none;解决。但是，在 chrome
更新到 27 版本之后就不可以用了。

2.还可以使用-webkit-transform:scale(0.5);注意-webkit-transform:scale(0.75);
收缩的是整个 span 的大小，这时候，必须要将 span 转换成块元素，可以使用 display：block/inline-block/...；

（8）超链接访问过后 hover 样式就不出现了，被点击访问过的超链接样式不再具有 hover 和 active 了
解决方法：改变 CSS 属性的排列顺序 L-V-H-A

（9）怪异模式问题：漏写 DTD 声明，Firefox 仍然会按照标准模式来解析网页，但在 IE 中会触发怪异模
式。为避免怪异模式给我们带来不必要的麻烦，最好养成书写 DTD 声明的好习惯。

2.18.li 与 li 之间有看不见的空白间隔是什么原因引起的？有什么解决办法？

浏览器会把 inline 元素间的空白字符（空格、换行、Tab 等）渲染成一个空格。而为了美观。我们通常是一个<li>放在一行，
这导致<li>换行后产生换行字符，它变成一个空格，占用了一个字符的宽度。

解决办法：

（1）为<li>设置 float:left。不足：有些容器是不能设置浮动，如左右切换的焦点图等。

（2）将所有<li>写在同一行。不足：代码不美观。

（3）将<ul>内的字符尺寸直接设为 0，即 font-size:0。不足：<ul>中的其他字符尺寸也被设为 0，需要额外重新设定其他
字符尺寸，且在 Safari 浏览器依然会出现空白间隔。

（4）消除<ul>的字符间隔 letter-spacing:-8px，不足：这也设置了<li>内的字符间隔，因此需要将<li>内的字符
间隔设为默认 letter-spacing:normal。
详细资料可以参考： 《li 与 li 之间有看不见的空白间隔是什么原因引起的？》

2.19.为什么要初始化 CSS 样式？

-因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对 CSS 初始化往往会出现浏览器之间的页面显示差异。

-当然，初始化样式会对 SEO 有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。

最简单的初始化方法：*{padding:0;margin:0;}（强烈不建议）

淘宝的样式初始化代码：

```
body,h1,h2,h3,h4,h5,h6,hr,p,blockquote,dl,dt,dd,ul,ol,li,pre,form,fieldset,legend
,button,input,textarea,th,td{margin:0;padding:0;}
body,button,input,select,textarea{font:12px/1.5tahoma,arial,\5b8b\4f53;}
h1,h2,h3,h4,h5,h6{font-size:100%;}
address,cite,dfn,em,var{font-style:normal;}
code,kbd,pre,samp{font-family:couriernew,courier,monospace;}
small{font-size:12px;}
ul,ol{list-style:none;}
```

```
a{text-decoration:none;}
a:hover{text-decoration:underline;}
sup{vertical-align:text-top;}
sub{vertical-align:text-bottom;}
legend{color:#000;}
fieldset,img{border:0;}
button,input,select,textarea{font-size:100%;}
table{border-collapse:collapse;border-spacing:0;}
```

2.20.什么是包含块，对于包含块的理解?

包含块（containingblock）就是元素用来计算和定位的一个框。

（1）根元素（很多场景下可以看成是<html>）被称为“初始包含块”，其尺寸等同于浏览器可视窗口的大小。

（2）对于其他元素，如果该元素的 position 是 relative 或者 static，则“包含块”由其最近的块容器祖先盒的 contentbox
边界形成。

（3）如果元素 position:fixed，则“包含块”是“初始包含块”。

（4）如果元素 position:absolute，则“包含块”由最近的 position 不为 static 的祖先元素建立，具体方式如下：

如果该祖先元素是纯 inline 元素，则规则略复杂：
•假设给内联元素的前后各生成一个宽度为 0 的内联盒子（inlinebox），则这两个内联盒子的 paddingbox 外面的包
围盒就是内联元素的“包含块”；
•如果该内联元素被跨行分割了，那么“包含块”是未定义的，也就是 CSS2.1 规范并没有明确定义，浏览器自行发挥
否则，“包含块”由该祖先的 paddingbox 边界形成。

如果没有符合条件的祖先元素，则“包含块”是“初始包含块”。

2.21.CSS 里的 visibility 属性有个 collapse 属性值是干嘛用的？在不同浏览器下以后什么区别？

（1）对于一般的元素，它的表现跟 visibility：hidden;是一样的。元素是不可见的，但此时仍占用页面空间。

（2）但例外的是，如果这个元素是 table 相关的元素，例如 table 行，tablegroup，table 列，tablecolumngroup，它的
表现却跟 display:none 一样，也就是说，它们占用的空间也会释放。

在不同浏览器下的区别：

在谷歌浏览器里，使用 collapse 值和使用 hidden 值没有什么区别。

在火狐浏览器、Opera 和 IE11 里，使用 collapse 值的效果就如它的字面意思：table 的行会消失，它的下面一行会补充它的位
置。

详细资料可以参考：《CSS 里的 visibility 属性有个鲜为人知的属性值：collapse》

2.22.width:auto 和 width:100%的区别

一般而言

width:100%会使元素 box 的宽度等于父元素的 contentbox 的宽度。

width:auto 会使元素撑满整个父元素，margin、border、padding、content 区域会自动分配水平空间。
23.绝对定位元素与非绝对定位元素的百分比计算的区别
绝对定位元素的宽高百分比是相对于临近的 position 不为 static 的祖先元素的 paddingbox 来计算的。

非绝对定位元素的宽高百分比则是相对于父元素的 contentbox 来计算的。

2.24.简单介绍使用图片 base64 编码的优点和缺点。

base64 编码是一种图片处理格式，通过特定的算法将图片编码成一长串字符串，在页面上显示的时候，可以用该字符串来代替图片的
url 属性。

使用 base64 的优点是：

（1）减少一个图片的 HTTP 请求

使用 base64 的缺点是：

（1）根据 base64 的编码原理，编码后的大小会比原文件大小大 1/3，如果把大图片编码到 html/css 中，不仅会造成文件体
积的增加，影响文件的加载速度，还会增加浏览器对 html 或 css 文件解析渲染的时间。

（2）使用 base64 无法直接缓存，要缓存只能缓存包含 base64 的文件，比如 HTML 或者 CSS，这相比域直接缓存图片的效果要
差很多。

（3）兼容性的问题，ie8 以前的浏览器不支持。

一般一些网站的小图标可以使用 base64 图片来引入。
详细资料可以参考：《玩转图片 base64 编码》《前端开发中，使用 base64 图片的弊端是什么？》《小 tip:base64:URL 背景图片与 web 页面性能优化》

2.25.'display'、'position'和'float'的相互关系？

（1）首先我们判断 display 属性是否为 none，如果为 none，则 position 和 float 属性的值不影响元素最后的表现。

（2）然后判断 position 的值是否为 absolute 或者 fixed，如果是，则 float 属性失效，并且 display 的值应该被
设置为 table 或者 block，具体转换需要看初始转换值。

（3）如果 position 的值不为 absolute 或者 fixed，则判断 float 属性的值是否为 none，如果不是，则 display
的值则按上面的规则转换。注意，如果 position 的值为 relative 并且 float 属性的值存在，则 relative 相对
于浮动后的最终位置定位。

（4）如果 float 的值为 none，则判断元素是否为根元素，如果是根元素则 display 属性按照上面的规则转换，如果不是，
则保持指定的 display 属性值不变。

总的来说，可以把它看作是一个类似优先级的机制，"position:absolute"和"position:fixed"优先级最高，有它存在
的时候，浮动不起作用，'display'的值也需要调整；其次，元素的'float'特性的值不是"none"的时候或者它是根元素
的时候，调整'display'的值；最后，非根元素，并且非浮动元素，并且非绝对定位的元素，'display'特性值同设置值。

详细资料可以参考：《position 跟 display、margincollapse、overflow、float 这些特性相互叠加后会怎么样？》

2.26.margin 重叠问题的理解。

相关知识点：
块级元素的上外边距（margin-top）与下外边距（margin-bottom）有时会合并为单个外边距，这样的现象称为“margin 合
并”。

产生折叠的必备条件：margin 必须是邻接的!

而根据 w3c 规范，两个 margin 是邻接的必须满足以下条件：

•必须是处于常规文档流（非 float 和绝对定位）的块级盒子，并且处于同一个 BFC 当中。
•没有线盒，没有空隙，没有 padding 和 border 将他们分隔开
•都属于垂直方向上相邻的外边距，可以是下面任意一种情况
•元素的 margin-top 与其第一个常规文档流的子元素的 margin-top
•元素的 margin-bottom 与其下一个常规文档流的兄弟元素的 margin-top
•height 为 auto 的元素的 margin-bottom 与其最后一个常规文档流的子元素的 margin-bottom
•高度为 0 并且最小高度也为 0，不包含常规文档流的子元素，并且自身没有建立新的 BFC 的元素的 margin-top
和 margin-bottom

margin 合并的 3 种场景：

（1）相邻兄弟元素 margin 合并。

解决办法：
•设置块状格式化上下文元素（BFC）
（2）父级和第一个/最后一个子元素的 margin 合并。
解决办法：

对于 margin-top 合并，可以进行如下操作（满足一个条件即可）：
•父元素设置为块状格式化上下文元素；
•父元素设置 border-top 值；
•父元素设置 padding-top 值；
•父元素和第一个子元素之间添加内联元素进行分隔。

对于 margin-bottom 合并，可以进行如下操作（满足一个条件即可）：
•父元素设置为块状格式化上下文元素；
•父元素设置 border-bottom 值；
•父元素设置 padding-bottom 值；
•父元素和最后一个子元素之间添加内联元素进行分隔；
•父元素设置 height、min-height 或 max-height。

（3）空块级元素的 margin 合并。

解决办法：
•设置垂直方向的 border；
•设置垂直方向的 padding；
•里面添加内联元素（直接 Space 键空格是没用的）；
•设置 height 或者 min-height。
回答：

margin 重叠指的是在垂直方向上，两个相邻元素的 margin 发生重叠的情况。

一般来说可以分为四种情形：

第一种是相邻兄弟元素的 marin-bottom 和 margin-top 的值发生重叠。这种情况下我们可以通过设置其中一个元素为 BFC
来解决。

第二种是父元素的 margin-top 和子元素的 margin-top 发生重叠。它们发生重叠是因为它们是相邻的，所以我们可以通过这
一点来解决这个问题。我们可以为父元素设置 border-top、padding-top 值来分隔它们，当然我们也可以将父元素设置为 BFC
来解决。

第三种是高度为 auto 的父元素的 margin-bottom 和子元素的 margin-bottom 发生重叠。它们发生重叠一个是因为它们相
邻，一个是因为父元素的高度不固定。因此我们可以为父元素设置 border-bottom、padding-bottom 来分隔它们，也可以为
父元素设置一个高度，max-height 和 min-height 也能解决这个问题。当然将父元素设置为 BFC 是最简单的方法。

第四种情况，是没有内容的元素，自身的 margin-top 和 margin-bottom 发生的重叠。我们可以通过为其设置 border、pa
dding 或者高度来解决这个问题。

2.27.对 BFC 规范（块级格式化上下文：blockformattingcontext）的理解？

相关知识点：
块格式化上下文（BlockFormattingContext，BFC）是 Web 页面的可视化 CSS 渲染的一部分，是布局过程中生成块级盒
子的区域，也是浮动元素与其他元素的交互限定区域。

通俗来讲

•BFC 是一个独立的布局环境，可以理解为一个容器，在这个容器中按照一定规则进行物品摆放，并且不会影响其它环境中的物品。
•如果一个元素符合触发 BFC 的条件，则 BFC 中的元素布局不受外部影响。

创建 BFC

（1）根元素或包含根元素的元素
（2）浮动元素 float＝left|right 或 inherit（≠none）
（3）绝对定位元素 position＝absolute 或 fixed
（4）display＝inline-block|flex|inline-flex|table-cell 或 table-caption

（5）overflow＝hidden|auto 或 scroll(≠visible)

回答：
BFC 指的是块级格式化上下文，一个元素形成了 BFC 之后，那么它内部元素产生的布局不会影响到外部元素，外部元素的布局也
不会影响到 BFC 中的内部元素。一个 BFC 就像是一个隔离区域，和其他区域互不影响。

一般来说根元素是一个 BFC 区域，浮动和绝对定位的元素也会形成 BFC，display 属性的值为 inline-block、flex 这些
属性时也会创建 BFC。还有就是元素的 overflow 的值不为 visible 时都会创建 BFC。
详细资料可以参考：《深入理解 BFC 和 MarginCollapse》《前端面试题-BFC（块格式化上下文）》

2.28.IFC 是什么？

IFC 指的是行级格式化上下文，它有这样的一些布局规则：

（1）行级上下文内部的盒子会在水平方向，一个接一个地放置。
（2）当一行不够的时候会自动切换到下一行。
（3）行级上下文的高度由内部最高的内联盒子的高度决定。
详细资料可以参考：《[译]:BFC 与 IFC》《BFC 和 IFC 的理解（布局）》

2.29.请解释一下为什么需要清除浮动？清除浮动的方式

浮动元素可以左右移动，直到遇到另一个浮动元素或者遇到它外边缘的包含框。浮动框不属于文档流中的普通流，当元素浮动之后，
不会影响块级元素的布局，只会影响内联元素布局。此时文档流中的普通流就会表现得该浮动框不存在一样的布局模式。当包含框
的高度小于浮动框的时候，此时就会出现“高度塌陷”。

清除浮动是为了清除使用浮动元素产生的影响。浮动的元素，高度会塌陷，而高度的塌陷使我们页面后面的布局不能正常显示。

清除浮动的方式

（1）使用 clear 属性清除浮动。参考 28。

（2）使用 BFC 块级格式化上下文来清除浮动。参考 26。

因为 BFC 元素不会影响外部元素的特点，所以 BFC 元素也可以用来清除浮动的影响，因为如果不清除，子元素浮动则父元
素高度塌陷，必然会影响后面元素布局和定位，这显然有违 BFC 元素的子元素不会影响外部元素的设定。

2.30.使用 clear 属性清除浮动的原理？

使用 clear 属性清除浮动，其语法如下：

clear:none|left|right|both

如果单看字面意思，clear:left 应该是“清除左浮动”，clear:right 应该是“清除右浮动”的意思，实际上，这种解释是有问
题的，因为浮动一直还在，并没有清除。

官方对 clear 属性的解释是：“元素盒子的边不能和前面的浮动元素相邻。”，我们对元素设置 clear 属性是为了避免浮动元素
对该元素的影响，而不是清除掉浮动。

还需要注意的一点是 clear 属性指的是元素盒子的边不能和前面的浮动元素相邻，注意这里“前面的”3 个字，也就是 clear 属
性对“后面的”浮动元素是不闻不问的。考虑到 float 属性要么是 left，要么是 right，不可能同时存在，同时由于 clear
属性对“后面的”浮动元素不闻不问，因此，当 clear:left 有效的时候，clear:right 必定无效，也就是此时 clear:left
等同于设置 clear:both；同样地，clear:right 如果有效也是等同于设置 clear:both。由此可见，clear:left 和 cle
ar:right 这两个声明就没有任何使用的价值，至少在 CSS 世界中是如此，直接使用 clear:both 吧。

一般使用伪元素的方式清除浮动

```
.clear::after{
content:'';
display:table;//也可以是'block'，或者是'list-item'
clear:both;
}
```

clear 属性只有块级元素才有效的，而::after 等伪元素默认都是内联水平，这就是借助伪元素清除浮动影响时需要设置 disp
lay 属性值的原因。

2.31.zoom:1 的清除浮动原理?

清除浮动，触发 hasLayout；
zoom 属性是 IE 浏览器的专有属性，它可以设置或检索对象的缩放比例。解决 ie 下比较奇葩的 bug。譬如外边距（margin）
的重叠，浮动清除，触发 ie 的 haslayout 属性等。

来龙去脉大概如下：
当设置了 zoom 的值之后，所设置的元素就会就会扩大或者缩小，高度宽度就会重新计算了，这里一旦改变 zoom 值时其实也会发
生重新渲染，运用这个原理，也就解决了 ie 下子元素浮动时候父元素不随着自动扩大的问题。

zoom 属性是 IE 浏览器的专有属性，火狐和老版本的 webkit 核心的浏览器都不支持这个属性。然而，zoom 现在已经被逐步标
准化，出现在 CSS3.0 规范草案中。

目前非 ie 由于不支持这个属性，它们又是通过什么属性来实现元素的缩放呢？可以通过 css3 里面的动画属性 scale 进行缩放。

2.32.移动端的布局用过媒体查询吗？

假设你现在正用一台显示设备来阅读这篇文章，同时你也想把它投影到屏幕上，或者打印出来，而显示设备、屏幕投影和打印等这些
媒介都有自己的特点，CSS 就是为文档提供在不同媒介上展示的适配方法

当媒体查询为真时，相关的样式表或样式规则会按照正常的级联规被应用。当媒体查询返回假，标签上带有媒体查询的样式表仍将被
下载（只不过不会被应用）。

包含了一个媒体类型和至少一个使用宽度、高度和颜色等媒体属性来限制样式表范围的表达式。CSS3 加入的媒体查询使得无需修改
内容便可以使样式应用于某些特定的设备范围。
详细资料可以参考：《CSS3@media 查询》《响应式布局和自适应布局详解》

2.33.使用 CSS 预处理器吗？喜欢哪个？

SASS（SASS、LESS 没有本质区别，只因为团队前端都是用的 SASS）

2.34.CSS 优化、提高性能的方法有哪些？

加载性能：

（1）css 压缩：将写好的 css 进行打包压缩，可以减少很多的体积。
（2）css 单一样式：当需要下边距和左边距的时候，很多时候选择:margin:top0bottom0;但 margin-bottom:bot
tom;margin-left:left;执行的效率更高。
（3）减少使用@import,而建议使用 link，因为后者在页面加载时一起加载，前者是等待页面加载完成之后再进行加载。

选择器性能：

（1）关键选择器（keyselector）。选择器的最后面的部分为关键选择器（即用来匹配目标元素的部分）。CSS 选择符是从右到
左进行匹配的。当使用后代选择器的时候，浏览器会遍历所有子元素来确定是否是指定的元素等等；

（2）如果规则拥有 ID 选择器作为其关键选择器，则不要为规则增加标签。过滤掉无关的规则（这样样式系统就不会浪费时间去匹
配它们了）。

（3）避免使用通配规则，如*{}计算次数惊人！只对需要用到的元素进行选择。

（4）尽量少的去对标签进行选择，而是用 class。

（5）尽量少的去使用后代选择器，降低选择器的权重值。后代选择器的开销是最高的，尽量将选择器的深度降到最低，最高不要超过
三层，更多的使用类来关联每一个标签元素。

（6）了解哪些属性是可以通过继承而来的，然后避免对这些属性重复指定规则。

渲染性能：

（1）慎重使用高性能属性：浮动、定位。

（2）尽量减少页面重排、重绘。

（3）去除空规则：{ }。空规则的产生原因一般来说是为了预留样式。去除这些空规则无疑能减少 css 文档体积。

（4）属性值为 0 时，不加单位。

（5）属性值为浮动小数 0.**，可以省略小数点之前的 0。

（6）标准化各种浏览器前缀：带浏览器前缀的在前。标准属性在后。

（7）不使用@import 前缀，它会影响 css 的加载速度。

（8）选择器优化嵌套，尽量避免层级过深。

（9）css 雪碧图，同一页面相近部分的小图标，方便使用，减少页面的请求次数，但是同时图片本身会变大，使用时，优劣考虑清
楚，再使用。

（10）正确使用 display 的属性，由于 display 的作用，某些样式组合会无效，徒增样式体积的同时也影响解析性能。

（11）不滥用 web 字体。对于中文网站来说 WebFonts 可能很陌生，国外却很流行。webfonts 通常体积庞大，而且一些浏
览器在下载 webfonts 时会阻塞页面渲染损伤性能。

可维护性、健壮性：

（1）将具有相同属性的样式抽离出来，整合并通过 class 在页面中进行使用，提高 css 的可维护性。
（2）样式与内容分离：将 css 代码定义到外部 css 中。
详细资料可以参考：《CSS 优化、提高性能的方法有哪些？》《CSS 优化，提高性能的方法》

2.35.浏览器是怎样解析 CSS 选择器的？

样式系统从关键选择器开始匹配，然后左移查找规则选择器的祖先元素。只要选择器的子树一直在工作，样式系统就会持续左移，直
到和规则匹配，或者是因为不匹配而放弃该规则。

试想一下，如果采用从左至右的方式读取 CSS 规则，那么大多数规则读到最后（最右）才会发现是不匹配的，这样做会费时耗能，
最后有很多都是无用的；而如果采取从右向左的方式，那么只要发现最右边选择器不匹配，就可以直接舍弃了，避免了许多无效匹配。
详细资料可以参考：《探究 CSS 解析原理》

2.36.在网页中应该使用奇数还是偶数的字体？为什么呢？

（1）偶数字号相对更容易和 web 设计的其他部分构成比例关系。比如：当我用了 14px 的正文字号，我可能会在一些地方用 14
×0.5=7px 的 margin，在另一些地方用 14×1.5=21px 的标题字号。
（2）浏览器缘故，低版本的浏览器 ie6 会把奇数字体强制转化为偶数，即 13px 渲染为 14px。
（3）系统差别，早期的 Windows 里，中易宋体点阵只有 12 和 14、15、16px，唯独缺少 13px。
详细资料可以参考：《谈谈网页中使用奇数字体和偶数字体》《现在网页设计中的为什么少有人用 11px、13px、15px 等奇数的字体？》

2.37.margin 和 padding 分别适合什么场景使用？

margin 是用来隔开元素与元素的间距；padding 是用来隔开元素与内容的间隔。
margin 用于布局分开元素使元素与元素互不相干。
padding 用于元素与内容之间的间隔，让内容（文字）与（包裹）元素之间有一段距离。

何时应当使用 margin：

•需要在 border 外侧添加空白时。
•空白处不需要背景（色）时。
•上下相连的两个盒子之间的空白，需要相互抵消时。如 15px+20px 的 margin，将得到 20px 的空白。

何时应当时用 padding：
•需要在 border 内测添加空白时。
•空白处需要背景（色）时。
•上下相连的两个盒子之间的空白，希望等于两者之和时。如 15px+20px 的 padding，将得到 35px 的空白。

2.38.抽离样式模块怎么写，说出思路，有无实践经验？[阿里航旅的面试题]

我的理解是把常用的 css 样式单独做成 css 文件……通用的和业务相关的分离出来，通用的做成样式模块儿共享，业务相关的，放
进业务相关的库里面做成对应功能的模块儿。
详细资料可以参考： 《CSS 规范-分类方法》

2.39.简单说一下 css3 的 all 属性。

all 属性实际上是所有 CSS 属性的缩写，表示，所有的 CSS 属性都怎样怎样，但是，不包括 unicode-bidi 和 direction
这两个 CSS 属性。支持三个 CSS 通用属性值，initial,inherit,unset。

initial 是初始值的意思，也就是该元素元素都除了 unicode-bidi 和 direction 以外的 CSS 属性都使用属性的默认初始
值。

inherit 是继承的意思，也就是该元素除了 unicode-bidi 和 direction 以外的 CSS 属性都继承父元素的属性值。

unset 是取消设置的意思，也就是当前元素浏览器或用户设置的 CSS 忽略，然后如果是具有继承特性的 CSS，如 color，则
使用继承值；如果是没有继承特性的 CSS 属性，如 background-color，则使用初始值。

详细资料可以参考： 《简单了解 CSS3 的 all 属性》

2.40.为什么不建议使用统配符初始化 css 样式。

采用*{pading:0;margin:0;}这样的写法好处是写起来很简单，但是是通配符，需要把所有的标签都遍历一遍，当网站较大时，
样式比较多，这样写就大大的加强了网站运行的负载，会使网站加载的时候需要很长一段时间，因此一般大型的网站都有分层次的一
套初始化样式。

出于性能的考虑，并不是所有标签都会有 padding 和 margin，因此对常见的具有默认 padding 和 margin 的元素初始化即
可，并不需使用通配符*来初始化。

2.41.absolute 的 containingblock（包含块）计算方式跟正常流有什么不同？

（1）内联元素也可以作为“包含块”所在的元素；

（2）“包含块”所在的元素不是父块级元素，而是最近的 position 不为 static 的祖先元素或根元素；

（3）边界是 paddingbox 而不是 contentbox。

2.42.对于 hasLayout 的理解？

hasLayout 是 IE 特有的一个属性。很多的 IE 下的 cssbug 都与其息息相关。在 IE 中，一个元素要么自己对自身的内容进
行计算大小和组织，要么依赖于父元素来计算尺寸和组织内容。当一个元素的 hasLayout 属性值为 true 时，它负责对自己和可
能的子孙元素进行尺寸计算和定位。虽然这意味着这个元素需要花更多的代价来维护自身和里面的内容，而不是依赖于祖先元素来完
成这些工作。
详细资料可以参考：《CSS 基础篇--CSS 中 IE 浏览器的 hasLayout，IE 低版本的 bug 根源》《CSS 魔法堂：hasLayout 原来是这样的!》

2.43.元素竖向的百分比设定是相对于容器的高度吗？

如果是 height 的话，是相对于包含块的高度。

如果是 padding 或者 margin 竖直方向的属性则是相对于包含块的宽度。

2.44.全屏滚动的原理是什么？用到了 CSS 的哪些属性？（待深入实践）

原理：有点类似于轮播，整体的元素一直排列下去，假设有 5 个需要展示的全屏页面，那么高度是 500%，只是展示 100%，容器及容
器内的页面取当前可视区高度，同时容器的父级元素 overflow 属性值设为 hidden，通过更改容器可视区的位置来实现全
屏滚动效果。主要是响应鼠标事件，页面通过 CSS 的动画效果，进行移动。

overflow：hidden；transition：all1000msease；
详细资料可以参考：《js 实现网页全屏切换（平滑过渡），鼠标滚动切换》《用 ES6 写全屏滚动插件》

2.45.什么是响应式设计？响应式设计的基本原理是什么？如何兼容低版本的IE？（待深入了解）

响应式网站设计是一个网站能够兼容多个终端，而不是为每一个终端做一个特定的版本。基本原理是通过媒体查询检测不同的设备屏
幕尺寸做处理。页面头部必须有 meta 声明的 viewport。
详细资料可以参考：《响应式布局原理》《响应式布局的实现方法和原理》

2.46.视差滚动效果，如何给每页做不同的动画？（回到顶部，向下滑动要再次出现，和只出现一次分别怎么做？）

视差滚动是指多层背景以不同的速度移动，形成立体的运动效果，带来非常出色的视觉体验。
详细资料可以参考：《如何实现视差滚动效果的网页？》

2.47.如何修改 chrome 记住密码后自动填充表单的黄色背景？

chrome 表单自动填充后，input 文本框的背景会变成黄色的，通过审查元素可以看到这是由于 chrome 会默认给自动填充的 in
put 表单加上 input:-webkit-autofill 私有属性，然后对其赋予以下样式：

```
{
background-color:rgb(250,255,189)!important;
background-image:none!important;
color:rgb(0,0,0)!important;
}
```

对 chrome 默认定义的 background-color，background-image，color 使用 important 是不能提高其优先级的，但是
其他属性可使用。

使用足够大的纯色内阴影来覆盖 input 输入框的黄色背景，处理如下

```
input:-webkit-autofill,textarea:-webkit-autofill,select:-webkit-autofill{
-webkit-box-shadow:000px1000pxwhiteinset;
border:1pxsolid#CCC!important;
}
```

详细资料可以参考：《去掉 chrome 记住密码后的默认填充样式》《修改谷歌浏览器 chrome 记住密码后自动填充表单的黄色背景》

2.48.怎么让 Chrome 支持小于 12px 的文字？

在谷歌下 css 设置字体大小为 12px 及以下时，显示都是一样大小，都是默认 12px。

解决办法：

（1）可以使用 Webkit 的内核的-webkit-text-size-adjust 的私有 CSS 属性来解决，只要加了 -webkit-text-size
-adjust:none;字体大小就不受限制了。但是 chrome 更新到 27 版本之后就不可以用了。所以高版本 chrome 谷歌浏览器
已经不再支持-webkit-text-size-adjust 样式，所以要使用时候慎用。

（2）还可以使用 css3 的 transform 缩放属性 -webkit-transform:scale(0.5); 注意 -webkit-transform:scale(0.
75);收缩的是整个元素的大小，这时候，如果是内联元素，必须要将内联元素转换成块元素，可以使用 display：block/
inline-block/...；

（3）使用图片：如果是内容固定不变情况下，使用将小于 12px 文字内容切出做图片，这样不影响兼容也不影响美观。
详细资料可以参考：《谷歌浏览器不支持 CSS 设置小于 12px 的文字怎么办？》

2.49.让页面里的字体变清晰，变细用 CSS 怎么做？

webkit 内核的私有属性：-webkit-font-smoothing，用于字体抗锯齿，使用后字体看起来会更清晰舒服。

在 MacOS 测试环境下面设置-webkit-font-smoothing:antialiased;但是这个属性仅仅是面向 MacOS，其他操作系统设
置后无效。
详细资料可以参考：《让字体变的更清晰 CSS 中-webkit-font-smoothing》

2.50.font-style 属性中 italic 和 oblique 的区别？

italic 和 oblique 这两个关键字都表示“斜体”的意思。

它们的区别在于，italic 是使用当前字体的斜体字体，而 oblique 只是单纯地让文字倾斜。如果当前字体没有对应的斜体字体，
则退而求其次，解析为 oblique，也就是单纯形状倾斜。

2.51.设备像素、css 像素、设备独立像素、dpr、ppi 之间的区别？

设备像素指的是物理像素，一般手机的分辨率指的就是设备像素，一个设备的设备像素是不可变的。

css 像素和设备独立像素是等价的，不管在何种分辨率的设备上，css 像素的大小应该是一致

的，css 像素是一个相对单位，它是相
对于设备像素的，一个 css 像素的大小取决于页面缩放程度和 dpr 的大小。

dpr 指的是设备像素和设备独立像素的比值，一般的 pc 屏幕，dpr=1。在 iphone4 时，苹果推出了 retina 屏幕，它的 dpr
为 2。屏幕的缩放会改变 dpr 的值。

ppi 指的是每英寸的物理像素的密度，ppi 越大，屏幕的分辨率越大。
详细资料可以参考：《什么是物理像素、虚拟像素、逻辑像素、设备像素，什么又是 PPI,DPI,DPR 和 DIP》《前端工程师需要明白的「像素」》《CSS 像素、物理像素、逻辑像素、设备像素比、PPI、Viewport》《前端开发中像素的概念》

2.52.layoutviewport、visualviewport 和 idealviewport 的区别？

相关知识点：
如果把移动设备上浏览器的可视区域设为 viewport 的话，某些网站就会因为 viewport 太窄而显示错乱，所以这些浏览器就决定
默认情况下把 viewport 设为一个较宽的值，比如 980px，这样的话即使是那些为桌面设计的网站也能在移动浏览器上正常显示了。
ppk 把这个浏览器默认的 viewport 叫做 layoutviewport。

layoutviewport 的宽度是大于浏览器可视区域的宽度的，所以我们还需要一个 viewport 来代表浏览器可视区域的大小，ppk 把
这个 viewport 叫做 visualviewport。

idealviewport 是最适合移动设备的 viewport，idealviewport 的宽度等于移动设备的屏幕宽度，只要在 css 中把某一元
素的宽度设为 idealviewport 的宽度（单位用 px），那么这个元素的宽度就是设备屏幕的宽度了，也就是宽度为 100%的效果。i
dealviewport 的意义在于，无论在何种分辨率的屏幕下，那些针对 idealviewport 而设计的网站，不需要用户手动缩放，也
不需要出现横向滚动条，都可以完美的呈现给用户。
回答：
移动端一共需要理解三个 viewport 的概念的理解。

第一个视口是布局视口，在移动端显示网页时，由于移动端的屏幕尺寸比较小，如果网页使用移动端的屏幕尺寸进行布局的话，那么整
个页面的布局都会显示错乱。所以移动端浏览器提供了一个 layoutviewport 布局视口的概念，使用这个视口来对页面进行布局展
示，一般 layoutviewport 的大小为 980px，因此页面布局不会有太大的变化，我们可以通过拖动和缩放来查看到这个页面。

第二个视口指的是视觉视口，visualviewport 指的是移动设备上我们可见的区域的视口大小，一般为屏幕的分辨率的大小。visu

alviewport 和 layoutviewport 的关系，就像是我们通过窗户看外面的风景，视觉视口就是窗户，而外面的风景就是布局视口
中的网页内容。

第三个视口是理想视口，由于 layoutviewport 一般比 visualviewport 要大，所以想要看到整个页面必须通过拖动和缩放才
能实现。所以又提出了 idealviewport 的概念，idealviewport 下用户不用缩放和滚动条就能够查看到整个页面，并且页面在
不同分辨率下显示的内容大小相同。idealviewport 其实就是通过修改 layoutviewport 的大小，让它等于设备的宽度，这个
宽度可以理解为是设备独立像素，因此根据 idealviewport 设计的页面，在不同分辨率的屏幕下，显示应该相同。
详细资料可以参考：《移动前端开发之 viewport 的深入理解》《说说移动前端中 viewport（视口）》《移动端适配知识你到底知多少》

2.53.position:fixed;在 android 下无效怎么处理？

因为移动端浏览器默认的 viewport 叫做 layoutviewport。在移动端显示时，因为 layoutviewport 的宽度大于移动端屏幕
的宽度，所以页面会出现滚动条左右移动，fixed 的元素是相对 layoutviewport 来固定位置的，而不是移动端屏幕来固定位置的
，所以会出现感觉 fixed 无效的情况。

如果想实现 fixed 相对于屏幕的固定效果，我们需要改变的是 viewport 的大小为 idealviewport，可以如下设置：

```
<metaname="viewport"content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-sca
le=1.0,user-scalable=no"/>
```

2.54.如果需要手动写动画，你认为最小时间间隔是多久，为什么？（阿里）

多数显示器默认频率是 60Hz，即 1 秒刷新 60 次，所以理论上最小间隔为 1/60*1000ms＝16.7ms

2.55.如何让去除 inline-block 元素间间距？

移除空格、使用 margin 负值、使用 font-size:0、letter-spacing、word-spacing
详细资料可以参考： 《去除 inline-block 元素间间距的 N 种方法》

2.56.overflow:scroll 时不能平滑滚动的问题怎么处理？

以下代码可解决这种卡顿的问题：-webkit-overflow-scrolling:touch;是因为这行代码启用了硬件加速特性，所以滑动很流

畅。
详细资料可以参考：《解决页面使用 overflow:scroll 在 iOS 上滑动卡顿的问题》

2.57.有一个高度自适应的 div，里面有两个 div，一个高度 100px，希望另一个填满剩下的高度。

（1）外层 div 使用 position：relative；高度要求自适应的 div 使用 position:absolute;top:100px;bottom:0;
left:0;right:0;

（2）使用 flex 布局，设置主轴为竖轴，第二个 div 的 flex-grow 为 1。
详细资料可以参考：《有一个高度自适应的 div，里面有两个 div，一个高度 100px，希望另一个填满剩下的高度(三种方案)》

2.58.png、jpg、gif 这些图片格式解释一下，分别什么时候用。有没有了解过 webp？

相关知识点：

（1）BMP，是无损的、既支持索引色也支持直接色的、点阵图。这种图片格式几乎没有对数据进行压缩，所以 BMP 格式的图片通常
具有较大的文件大小。

（2）GIF 是无损的、采用索引色的、点阵图。采用 LZW 压缩算法进行编码。文件小，是 GIF 格式的优点，同时，GIF 格式还具
有支持动画以及透明的优点。但，GIF 格式仅支持 8bit 的索引色，所以 GIF 格式适用于对色彩要求不高同时需要文件体积
较小的场景。

（3）JPEG 是有损的、采用直接色的、点阵图。JPEG 的图片的优点，是采用了直接色，得益于更丰富的色彩，JPEG 非常适合用来
存储照片，与 GIF 相比，JPEG 不适合用来存储企业 Logo、线框类的图。因为有损压缩会导致图片模糊，而直接色的选用，
又会导致图片文件较 GIF 更大。

（4）PNG-8 是无损的、使用索引色的、点阵图。PNG 是一种比较新的图片格式，PNG-8 是非常好的 GIF 格式替代者，在可能的
情况下，应该尽可能的使用 PNG-8 而不是 GIF，因为在相同的图片效果下，PNG-8 具有更小的文件体积。除此之外，PNG-8
还支持透明度的调节，而 GIF 并不支持。现在，除非需要动画的支持，否则我们没有理由使用 GIF 而不是 PNG-8。

（5）PNG-24 是无损的、使用直接色的、点阵图。PNG-24 的优点在于，它压缩了图片的数据，使得同样效果的图片，PNG-24 格

式的文件大小要比 BMP 小得多。当然，PNG24 的图片还是要比 JPEG、GIF、PNG-8 大得多。

（6）SVG 是无损的、矢量图。SVG 是矢量图。这意味着 SVG 图片由直线和曲线以及绘制它们的方法组成。当你放大一个 SVG 图
片的时候，你看到的还是线和曲线，而不会出现像素点。这意味着 SVG 图片在放大时，不会失真，所以它非常适合用来绘制企
业 Logo、Icon 等。

（7）WebP 是谷歌开发的一种新图片格式，WebP 是同时支持有损和无损压缩的、使用直接色的、点阵图。从名字就可以看出来它是
为 Web 而生的，什么叫为 Web 而生呢？就是说相同质量的图片，WebP 具有更小的文件体积。现在网站上充满了大量的图片，
如果能够降低每一个图片的文件大小，那么将大大减少浏览器和服务器之间的数据传输量，进而降低访问延迟，提升访问体验。

•在无损压缩的情况下，相同质量的 WebP 图片，文件大小要比 PNG 小 26%；
•在有损压缩的情况下，具有相同图片精度的 WebP 图片，文件大小要比 JPEG 小 25%~34%；
•WebP 图片格式支持图片透明度，一个无损压缩的 WebP 图片，如果要支持透明度只需要 22%的格外文件大小。

但是目前只有 Chrome 浏览器和 Opera 浏览器支持 WebP 格式，兼容性不太好。
回答：
我了解到的一共有七种常见的图片的格式。

（1）第一种是 BMP 格式，它是无损压缩的，支持索引色和直接色的点阵图。由于它基本上没有进行压缩，因此它的文件体积一般比
较大。

（2）第二种是 GIF 格式，它是无损压缩的使用索引色的点阵图。由于使用了 LZW 压缩方法，因此文件的体积很小。并且 GIF 还
支持动画和透明度。但因为它使用的是索引色，所以它适用于一些对颜色要求不高且需要文件体积小的场景。

（3）第三种是 JPEG 格式，它是有损压缩的使用直接色的点阵图。由于使用了直接色，色彩较为丰富，一般适用于来存储照片。但
由于使用的是直接色，可能文件的体积相对于 GIF 格式来说更大。

（4）第四种是 PNG-8 格式，它是无损压缩的使用索引色的点阵图。它是 GIF 的一种很好的替代格式，它也支持透明度的调整，并
且文件的体积相对于 GIF 格式更小。一般来说如果不是需要动画的情况，我们都可以使用 PNG-8 格式代替 GIF 格式。

（5）第五种是 PNG-24 格式，它是无损压缩的使用直接色的点阵图。PNG-24 的优点是它使用了压缩算法，所以它的体积比 BMP

格式的文件要小得多，但是相对于其他的几种格式，还是要大一些。

（6）第六种格式是 svg 格式，它是矢量图，它记录的图片的绘制方式，因此对矢量图进行放大和缩小不会产生锯齿和失真。它一般
适合于用来制作一些网站 logo 或者图标之类的图片。

（7）第七种格式是 webp 格式，它是支持有损和无损两种压缩方式的使用直接色的点阵图。使用 webp 格式的最大的优点是，在相
同质量的文件下，它拥有更小的文件体积。因此它非常适合于网络图片的传输，因为图片体积的减少，意味着请求时间的减小，
这样会提高用户的体验。这是谷歌开发的一种新的图片格式，目前在兼容性上还不是太好。
详细资料可以参考：《图片格式那么多，哪种更适合你？》

2.59.浏览器如何判断是否支持 webp 格式图片

（1）宽高判断法。通过创建 image 对象，将其 src 属性设置为 webp 格式的图片，然后在 onload 事件中获取图片的宽高，如
果能够获取，则说明浏览器支持 webp 格式图片。如果不能获取或者触发了 onerror 函数，那么就说明浏览器不支持 webp 格
式的图片。

（2）canvas 判断方法。我们可以动态的创建一个 canvas 对象，通过 canvas 的 toDataURL 将设置为 webp 格式，然后判断
返回值中是否含有 image/webp 字段，如果包含则说明支持 WebP，反之则不支持。
详细资料可以参考：《判断浏览器是否支持 WebP 图片》《toDataURL()》

2.60.什么是 Cookie 隔离？（或者说：请求资源的时候不要让它带 cookie 怎么做）

网站向服务器请求的时候，会自动带上 cookie 这样增加表头信息量，使请求变慢。

如果静态文件都放在主域名下，那静态文件请求的时候都带有的 cookie 的数据提交给 server 的，非常浪费流量，所以不如隔离开
，静态资源放 CDN。

因为 cookie 有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有 cookie 数据，这样可以降低请
求头的大小，降低请求时间，从而达到降低整体请求延时的目的。

同时这种方式不会将 cookie 传入 WebServer，也减少了 WebServer 对 cookie 的处理分析环节，提高了 webserver 的
http 请求的解析速度。
详细资料可以参考：《CDN 是什么？使用 CDN 有什么优势？》

2.61.style 标签写在 body 后与 body 前有什么区别？

页面加载自上而下当然是先加载样式。写在 body 标签后由于浏览器以逐行方式对 HTML 文档进行解析，当解析到写在尾部的样式
表（外联或写在 style 标签）会导致浏览器停止之前的渲染，等待加载且解析样式表完成之后重新渲染，在 windows 的 IE 下可
能会出现 FOUC 现象（即样式失效导致的页面闪烁问题）

2.62.什么是 CSS 预处理器/后处理器？

CSS 预处理器定义了一种新的语言，其基本思想是，用一种专门的编程语言，为 CSS 增加了一些编程的特性，将 CSS 作为目标生成
文件，然后开发者就只要使用这种语言进行编码工作。通俗的说，CSS 预处理器用一种专门的编程语言，进行 Web 页面样式设计，然
后再编译成正常的 CSS 文件。

预处理器例如：LESS、Sass、Stylus，用来预编译 Sass 或 lesscsssprite，增强了 css 代码的复用性，还有层级、mixin、
变量、循环、函数等，具有很方便的 UI 组件模块化开发能力，极大的提高工作效率。

CSS 后处理器是对 CSS 进行处理，并最终生成 CSS 的预处理器，它属于广义上的 CSS 预处理器。我们很久以前就在用 CSS 后
处理器了，最典型的例子是 CSS 压缩工具（如 clean-css），只不过以前没单独拿出来说过。还有最近比较火的 Autoprefixer，
以 CanIUse 上的浏览器支持数据为基础，自动处理兼容性问题。

后处理器例如：PostCSS，通常被视为在完成的样式表中根据 CSS 规范处理 CSS，让其更有效；目前最常做的是给 CSS 属性添加浏
览器私有前缀，实现跨浏览器兼容性的问题。
详细资料可以参考：《CSS 预处理器和后处理器》

2.63.阐述一下 CSSSprites

将一个页面涉及到的所有图片都包含到一张大图中去，然后利用 CSS 的 background-image，background-repeat，background
-position 的组合进行背景定位。利用 CSSSprites 能很好地减少网页的 http 请求，从而很好的提高页面的性能；CSSSprites
能减少图片的字节。

优点：

减少 HTTP 请求数，极大地提高页面加载速度
增加图片信息重复度，提高压缩比，减少图片大小

更换风格方便，只需在一张或几张图片上修改颜色或样式即可实现

缺点：

图片合并麻烦
维护麻烦，修改一个图片可能需要重新布局整个图片，样式

2.64.使用 rem 布局的优缺点？

优点：
在屏幕分辨率千差万别的时代，只要将 rem 与屏幕分辨率关联起来就可以实现页面的整体缩放，使得在设备上的展现都统一起来了。
而且现在浏览器基本都已经支持 rem 了，兼容性也非常的好。

缺点：
（1）在奇葩的 dpr 设备上表现效果不太好，比如一些华为的高端机型用 rem 布局会出现错乱。
（2）使用 iframe 引用也会出现问题。
（3）rem 在多屏幕尺寸适配上与当前两大平台的设计哲学不一致。即大屏的出现到底是为了看得又大又清楚，还是为了看的更多的问
题。
详细资料可以参考：《css3 的字体大小单位 rem 到底好在哪？》《VW:是时候放弃 REM 布局了》《为什么设计稿是 750px》《使用 Flexible 实现手淘 H5 页面的终端适配》

2.65.几种常见的 CSS 布局

详细的资料可以参考：《几种常见的 CSS 布局》

2.66.画一条 0.5px 的线

采用 metaviewport 的方式

采用 border-image 的方式

采用 transform:scale()的方式
详细资料可以参考：《怎么画一条 0.5px 的边（更新）》

2.67.transition 和 animation 的区别

transition 关注的是 CSSproperty 的变化，property 值和时间的关系是一个三次贝塞尔曲线。

animation 作用于元素本身而不是样式属性，可以使用关键帧的概念，应该说可以实现更自由的动画效果。
详细资料可以参考：《CSSanimation 与 CSStransition 有何区别？》《CSS3Transition 和

Animation 区别及比较》《CSS 动画简介》《CSS 动画：animation、transition、transform、translate》

2.68.什么是首选最小宽度？

“首选最小宽度”，指的是元素最适合的最小宽度。

东亚文字（如中文）最小宽度为每个汉字的宽度。

西方文字最小宽度由特定的连续的英文字符单元决定。并不是所有的英文字符都会组成连续单元，一般会终止于空格（普通空格）、短
横线、问号以及其他非英文字符等。

如果想让英文字符和中文一样，每一个字符都用最小宽度单元，可以试试使用 CSS 中的 word-break:break-all。

2.69.为什么 height:100%会无效？

对于普通文档流中的元素，百分比高度值要想起作用，其父级必须有一个可以生效的高度值。

原因是如果包含块的高度没有显式指定（即高度由内容决定），并且该元素不是绝对定位，则计算值为 auto，因为解释成了 auto，
所以无法参与计算。

使用绝对定位的元素会有计算值，即使祖先元素的 height 计算为 auto 也是如此。

2.70.min-width/max-width 和 min-height/max-height 属性间的覆盖规则？

（1）max-width 会覆盖 width，即使 width 是行类样式或者设置了!important。

（2）min-width 会覆盖 max-width，此规则发生在 min-width 和 max-width 冲突的时候。

2.71.内联盒模型基本概念

（1）内容区域（contentarea）。内容区域指一种围绕文字看不见的盒子，其大小仅受字符本身特性控制，本质上是一个字符盒子
（characterbox）；但是有些元素，如图片这样的替换元素，其内容显然不是文字，不存在字符盒子之类的，因此，对于这些
元素，内容区域可以看成元素自身。

（2）内联盒子（inlinebox）。“内联盒子”不会让内容成块显示，而是排成一行，这里的“内联盒子”实际指的就是元素的“外在盒
子”，用来决定元素是内联还是块级。该盒子又可以细分为“内联盒子”和“匿名内联盒子”两

类。

（3）行框盒子（linebox），每一行就是一个“行框盒子”（实线框标注），每个“行框盒子”又是由一个一个“内联盒子”组成的。

（4）包含块（containingbox），由一行一行的“行框盒子”组成。

2.72.什么是幽灵空白节点？

“幽灵空白节点”是内联盒模型中非常重要的一个概念，具体指的是：在 HTML5 文档声明中，内联元素的所有解析和渲染表现就如同
每个行框盒子的前面有一个“空白节点”一样。这个“空白节点”永远透明，不占据任何宽度，看不见也无法通过脚本获取，就好像幽灵
一样，但又确确实实地存在，表现如同文本节点一样，因此，我称之为“幽灵空白节点”。

2.73.什么是替换元素？

通过修改某个属性值呈现的内容就可以被替换的元素就称为“替换元素”。因此，<img>、<object>、<video>、<iframe>或者表
单元素<textarea>和<input>和<select>都是典型的替换元素。

替换元素除了内容可替换这一特性以外，还有以下一些特性。

（1）内容的外观不受页面上的 CSS 的影响。用专业的话讲就是在样式表现在 CSS 作用域之外。如何更改替换元素本身的外观需要
类似 appearance 属性，或者浏览器自身暴露的一些样式接口，

（2）有自己的尺寸。在 Web 中，很多替换元素在没有明确尺寸设定的情况下，其默认的尺寸（不包括边框）是 300 像素×150 像
素，如<video>、<iframe>或者<canvas>等，也有少部分替换元素为 0 像素，如<img>图片，而表单元素的替换元素
的尺寸则和浏览器有关，没有明显的规律。

（3）在很多 CSS 属性上有自己的一套表现规则。比较具有代表性的就是 vertical-align 属性，对于替换元素和非替换元素，ve
rtical-align 属性值的解释是不一样的。比方说 vertical-align 的默认值的 baseline，很简单的属性值，基线之意，
被定义为字符 x 的下边缘，而替换元素的基线却被硬生生定义成了元素的下边缘。

（4）所有的替换元素都是内联水平元素，也就是替换元素和替换元素、替换元素和文字都是可以在一行显示的。但是，替换元素默认
的 display 值却是不一样的，有的是 inline，有的是 inline-block。

2.74.替换元素的计算规则？

替换元素的尺寸从内而外分为 3 类：固有尺寸、HTML 尺寸和 CSS 尺寸。

（1）固有尺寸指的是替换内容原本的尺寸。例如，图片、视频作为一个独立文件存在的时候，都是有着自己的宽度和高度的。

（2）HTML 尺寸只能通过 HTML 原生属性改变，这些 HTML 原生属性包括<img>的 width 和 height 属性、<input>的 s
ize 属性、<textarea>的 cols 和 rows 属性等。

（3）CSS 尺寸特指可以通过 CSS 的 width 和 height 或者 max-width/min-width 和 max-height/min-height 设置的
尺寸，对应盒尺寸中的 contentbox。

这 3 层结构的计算规则具体如下

（1）如果没有 CSS 尺寸和 HTML 尺寸，则使用固有尺寸作为最终的宽高。

（2）如果没有 CSS 尺寸，则使用 HTML 尺寸作为最终的宽高。

（3）如果有 CSS 尺寸，则最终尺寸由 CSS 属性决定。

（4）如果“固有尺寸”含有固有的宽高比例，同时仅设置了宽度或仅设置了高度，则元素依然按照固有的宽高比例显示。

（5）如果上面的条件都不符合，则最终宽度表现为 300 像素，高度为 150 像素。

（6）内联替换元素和块级替换元素使用上面同一套尺寸计算规则。

2.75.content 与替换元素的关系？

content 属性生成的对象称为“匿名替换元素”。

（1）我们使用 content 生成的文本是无法选中、无法复制的，好像设置了 userselect:none 声明一般，但是普通元素的文本
却可以被轻松选中。同时，content 生成的文本无法被屏幕阅读设备读取，也无法被搜索引擎抓取，因此，千万不要自以为是
地把重要的文本信息使用 content 属性生成，因为这对可访问性和 SEO 都很不友好。

（2）content 生成的内容不能左右:empty 伪类。

（3）content 动态生成值无法获取。

2.76.margin:auto 的填充规则？

margin 的'auto'可不是摆设，是具有强烈的计算意味的关键字，用来计算元素对应方向应该获得的剩余间距大小。但是触发 mar
gin:auto 计算有一个前提条件，就是 width 或 height 为 auto 时，元素是具有对应方向的自动填充特性的。

（1）如果一侧定值，一侧 auto，则 auto 为剩余空间大小。
（2）如果两侧均是 auto，则平分剩余空间。

2.77.margin 无效的情形

（1）display 计算值 inline 的非替换元素的垂直 margin 是无效的。对于内联替换元素，垂直 margin 有效，并且没有 ma
rgin 合并的问题。

（2）表格中的<tr>和<td>元素或者设置 display 计算值是 table-cell 或 table-row 的元素的 margin 都是无效的。

（3）绝对定位元素非定位方位的 margin 值“无效”。

（4）定高容器的子元素的 margin-bottom 或者宽度定死的子元素的 margin-right 的定位“失效”。

2.78.border 的特殊性？

（1）border-width 却不支持百分比。

（2）border-style 的默认值是 none，有一部分人可能会误以为是 solid。这也是单纯设置 border-width 或 border-col
or 没有边框显示的原因。

（3）border-style:double 的表现规则：双线宽度永远相等，中间间隔±1。

（4）border-color 默认颜色就是 color 色值。

（5）默认 background 背景图片是相对于 paddingbox 定位的。

2.79.什么是基线和 x-height？

字母 x 的下边缘（线）就是我们的基线。

x-height 指的就是小写字母 x 的高度，术语描述就是基线和等分线（meanline）（也称作中线，midline）之间的距离。在 C
SS 世界中，middle 指的是基线往上 1/2x-height 高度。我们可以近似理解为字母 x 交叉点那个位置。

ex 是 CSS 中的一个相对单位，指的是小写字母 x 的高度，没错，就是指 x-height。ex 的价值就在其副业上不受字体和字号影
响的内联元素的垂直居中对齐效果。内联元素默认是基线对齐的，而基线就是 x 的底部，而 1ex 就是一个 x 的高度。

2.80.line-height 的特殊性？

（1）对于非替换元素的纯内联元素，其可视高度完全由 line-height 决定。对于文本这样的纯内联元素，line-height 就是高
度计算的基石，用专业说法就是指定了用来计算行框盒子高度的基础高度。

（2）内联元素的高度由固定高度和不固定高度组成，这个不固定的部分就是这里的“行距”。换句话说，line-height 之所以起作
用，就是通过改变“行距”来实现的。在 CSS 中，“行距”分散在当前文字的上方和下方，也就是即使是第一行文字，其上方也是
有“行距”的，只不过这个“行距”的高度仅仅是完整“行距”高度的一半，因此，也被称为“半行距”。

（3）行距=line-height-font-size。

（4）border 以及 line-height 等传统 CSS 属性并没有小数像素的概念。如果标注的是文字上边距，则向下取整；如果是文字下
边距，则向上取整。

（5）对于纯文本元素，line-height 直接决定了最终的高度。但是，如果同时有替换元素，则 line-height 只能决定最小高度。

（6）对于块级元素，line-height 对其本身是没有任何作用的，我们平时改变 line-height，块级元素的高度跟着变化实际上是
通过改变块级元素里面内联级别元素占据的高度实现的。

（7）line-height 的默认值是 normal，还支持数值、百分比值以及长度值。为数值类型时，其最终的计算值是和当前 font-si
ze 相乘后的值。为百分比值时，其最终的计算值是和当前 font-size 相乘后的值。为长度值时原意不变。

（8）如果使用数值作为 line-height 的属性值，那么所有的子元素继承的都是这个值；但是，如果使用百分比值或者长度值作为
属性值，那么所有的子元素继承的是最终的计算值。

（9）无论内联元素 line-height 如何设置，最终父级元素的高度都是由数值大的那个 line-height 决定的。

（10）只要有“内联盒子”在，就一定会有“行框盒子”，就是每一行内联元素外面包裹的一层看不见的盒子。然后，重点来了，在每个
“行框盒子”前面有一个宽度为 0 的具有该元素的字体和行高属性的看不见的“幽灵空白节点”。

2.81.vertical-align 的特殊性？

（1）vertical-align 的默认值是 baseline，即基线对齐，而基线的定义是字母 x 的下边缘。因此，内联元素默认都是沿着字
母 x 的下边缘对齐的。对于图片等替换元素，往往使用元素本身的下边缘作为基线。：一个 inline-block 元素，如果里面
没有内联元素，或者 overflow 不是 visible，则该元素的基线就是其 margin 底边缘；否则其基线就是元素里面最后一行
内联元素的基线。

（2）vertical-align:top 就是垂直上边缘对齐，如果是内联元素，则和这一行位置最高的内联元素的顶部对齐；如果 display
计算值是 table-cell 的元素，我们不妨脑补成<td>元素，则和<tr>元素上边缘对齐。

（3）vertical-align:middle 是中间对齐，对于内联元素，元素的垂直中心点和行框盒子基线往上 1/2x-height 处对齐。对
于 table-cell 元素，单元格填充盒子相对于外面的表格行居中对齐。

（4）vertical-align 支持数值属性，根据数值的不同，相对于基线往上或往下偏移，如果是负值，往下偏移，如果是正值，往上
偏移。

（5）vertical-align 属性的百分比值则是相对于 line-height 的计算值计算的。

（6）vertical-align 起作用是有前提条件的，这个前提条件就是：只能应用于内联元素以及 display 值为 table-cell 的元
素。

（7）table-cell 元素设置 vertical-align 垂直对齐的是子元素，但是其作用的并不是子元素，而是 table-cell 元素自身。

2.82.overflow 的特殊性？

（1）一个设置了 overflow:hidden 声明的元素，假设同时存在 border 属性和 padding 属性，则当子元素内容超出容器宽度
高度限制的时候，剪裁的边界是 borderbox 的内边缘，而非 paddingbox 的内边缘。

（2）HTML 中有两个标签是默认可以产生滚动条的，一个是根元素<html>，另一个是文本域<textarea>。

（3）滚动条会占用容器的可用宽度或高度。

（4）元素设置了 overflow:hidden 声明，里面内容高度溢出的时候，滚动依然存在，仅仅滚动条不存在！

2.83.无依赖绝对定位是什么？

没有设置 left/top/right/bottom 属性值的绝对定位称为“无依赖绝对定位”。

无依赖绝对定位其定位的位置和没有设置 position:absolute 时候的位置相关。

2.84.absolute 与 overflow 的关系？

（1）如果 overflow 不是定位元素，同时绝对定位元素和 overflow 容器之间也没有定位元素，则 overflow 无法对 absolute
元素进行剪裁。

（2）如果 overflow 的属性值不是 hidden 而是 auto 或者 scroll，即使绝对定位元素高宽比 overflow 元素高宽还要大，也
都不会出现滚动条。

（3）overflow 元素自身 transform 的时候，Chrome 和 Opera 浏览器下的 overflow 剪裁是无效的。

2.85.clip 裁剪是什么？

所谓“可访问性隐藏”，指的是虽然内容肉眼看不见，但是其他辅助设备却能够进行识别和访问的隐藏。

clip 剪裁被我称为“最佳可访问性隐藏”的另外一个原因就是，它具有更强的普遍适应性，任何元素、任何场景都可以无障碍使用。

2.86.relative 的特殊性？

（1）相对定位元素的 left/top/right/bottom 的百分比值是相对于包含块计算的，而不是自身。注意，虽然定位位移是相对自身，但是百分比值的计算值不是。

（2）top 和 bottom 这两个垂直方向的百分比值计算跟 height 的百分比值是一样的，都是相对高度计算的。同时，如果包含块的高度是 auto，那么计算值是 0，偏移无效，也就是说，如果父元素没有设定高度或者不是“格式化高度”，那么 relative 类似 top:20%的代码等同于 top:0。

（3）当相对定位元素同时应用对立方向定位值的时候，也就是 top/bottom 和 left/right 同时使用的时候，只有一个方向的定位属性会起作用。而谁起作用则是与文档流的顺序有关的，

默认的文档流是自上而下、从左往右，因此 top/bottom 同时使用的时候，bottom 失效；left/right 同时使用的时候，right 失效。

2.87.什么是层叠上下文？

层叠上下文，英文称作 stackingcontext，是 HTML 中的一个三维的概念。如果一个元素含有层叠上下文，我们可以理解为这个元
素在 z 轴上就“高人一等”。

层叠上下文元素有如下特性：

（1）层叠上下文的层叠水平要比普通元素高（原因后面会说明）。
（2）层叠上下文可以阻断元素的混合模式。
（3）层叠上下文可以嵌套，内部层叠上下文及其所有子元素均受制于外部的“层叠上下文”。
（4）每个层叠上下文和兄弟元素独立，也就是说，当进行层叠变化或渲染的时候，只需要考虑后代元素。
（5）每个层叠上下文是自成体系的，当元素发生层叠的时候，整个元素被认为是在父层叠上下文的层叠顺序中。

层叠上下文的创建：

（1）页面根元素天生具有层叠上下文，称为根层叠上下文。根层叠上下文指的是页面根元素，可以看成是<html>元素。因此，页面中所有的元素一定处于至少一个“层叠结界”中。

（2）对于 position 值为 relative/absolute 以及 Firefox/IE 浏览器（不包括 Chrome 浏览器）下含有 position:fixed 声明的定位元素，当其 z-index 值不是 auto 的时候，会创建层叠上下文。Chrome 等 WebKit 内核浏览器下，position:fixed 元素天然层叠上下文元素，无须 z-index 为数值。根据我的测试，目前 IE 和 Firefox 仍是老套路。

（3）其他一些 CSS3 属性，比如元素的 opacity 值不是 1。

2.88.什么是层叠水平？

层叠水平，英文称作 stackinglevel，决定了同一个层叠上下文中元素在 z 轴上的显示顺序。

显而易见，所有的元素都有层叠水平，包括层叠上下文元素，也包括普通元素。然而，对普通元素的层叠水平探讨只局限在当前层叠上
下文元素中。

2.89.元素的层叠顺序？

层叠顺序，英文称作 stackingorder，表示元素发生层叠时有着特定的垂直显示顺序。

2.90.层叠准则？

（1）谁大谁上：当具有明显的层叠水平标识的时候，如生效的 z-index 属性值，在同一个层叠上下文领域，层叠水平值大的那一个覆盖小的那一个。

（2）后来居上：当元素的层叠水平一致、层叠顺序相同的时候，在 DOM 流中处于后面的元素会覆盖前面的元素。

2.91.font-weight 的特殊性？

如果使用数值作为 font-weight 属性值，必须是 100～900 的整百数。因为这里的数值仅仅是外表长得像数值，实际上是一个具有特定含义的关键字，并且这里的数值关键字和字母关键字之间是有对应关系的。

2.92.text-indent 的特殊性？

（1）text-indent 仅对第一行内联盒子内容有效。

（2）非替换元素以外的 display 计算值为 inline 的内联元素设置 text-indent 值无效，如果计算值 inline-block/inli
ne-table 则会生效。

（3）<input>标签按钮 text-indent 值无效。

（4）<button>标签按钮 text-indent 值有效。

（5）text-indent 的百分比值是相对于当前元素的“包含块”计算的，而不是当前元素。

2.93.letter-spacing 与字符间距？

letter-spacing 可以用来控制字符之间的间距，这里说的“字符”包括英文字母、汉字以及空格等。

letter-spacing 具有以下一些特性。

（1）继承性。
（2）默认值是 normal 而不是 0。虽然说正常情况下，normal 的计算值就是 0，但两者还是有差别的，在有些场景下，letter-spacing 会调整 normal 的计算值以实现更好的版面布局。
（3）支持负值，且值足够大的时候，会让字符形成重叠，甚至反向排列。
（4）和 text-indent 属性一样，无论值多大或多小，第一行一定会保留至少一个字符。
（5）支持小数值，即使 0.1px 也是支持的。

（6）暂不支持百分比值。

2.94.word-spacing 与单词间距？

letter-spacing 作用于所有字符，但 word-spacing 仅作用于空格字符。换句话说，word-spacing 的作用就是增加空格的间隙
宽度。

2.95.white-space 与换行和空格的控制？

white-space 属性声明了如何处理元素内的空白字符，这类空白字符包括 Space（空格）键、Enter（回车）键、Tab（制表符）
键产生的空白。因此，white-space 可以决定图文内容是否在一行显示（回车空格是否生效），是否显示大段连续空白（空格是否
生效）等。

其属性值包括下面这些。
•normal：合并空白字符和换行符。
•pre：空白字符不合并，并且内容只在有换行符的地方换行。
•nowrap：该值和 normal 一样会合并空白字符，但不允许文本环绕。
•pre-wrap：空白字符不合并，并且内容只在有换行符的地方换行，同时允许文本环绕。
•pre-line：合并空白字符，但只在有换行符的地方换行，允许文本环绕。

2.96.隐藏元素的 background-image 到底加不加载？

相关知识点：
根据测试，一个元素如果 display 计算值为 none，在 IE 浏览器下（IE8～IE11，更高版本不确定）依然会发送图片请求，Fire
fox 浏览器不会，至于 Chrome 和 Safari 浏览器则似乎更加智能一点：如果隐藏元素同时又设置了 background-image，则图片
依然会去加载；如果是父元素的 display 计算值为 none，则背景图不会请求，此时浏览器或许放心地认为这个背景图暂时是不会使
用的。

如果不是 background-image，而是<img>元素，则设置 display:none 在所有浏览器下依旧都会请求图片资源。

还需要注意的是如果设置的样式没有对应的元素，则 background-image 也不会加载。hover 情况下的 background-image，在触
发时加载。
回答：
-（1）元素的背景图片
-元素本身设置 display:none，会请求图片 -父级元素设置 display:none，不会请求图片 -样

式没有元素使用，不会请求 -:hover 样式下，触发时请求
-（2）img 标签图片任何情况下都会请求图片
详细资料可以参考： 《CSS 控制前端图片 HTTP 请求的各种情况示例》

2.97.如何实现单行 / 多行文本溢出的省略（...）？

```
/*单行文本溢出*/p {
   overflow: hidden;
   text-overflow: ellipsis;
   white-space: nowrap;
}
/*多行文本溢出*/p {
   position: relative;
   line-height: 1.5em;
   /*高度为需要显示的行数*行高，比如这里我们显示两行，则为 3*/
   height: 3em;
   overflow: hidden;
}
p:after {
   content: "...";
   position: absolute;
   bottom: 0;
   right: 0;
   background-color: #fff;
}
```

详细资料可以参考： 《【CSS/JS】如何实现单行 / 多行文本溢出的省略》 《CSS 多行文本溢出省略显示》

2.98.常见的元素隐藏方式？

-（1）使用 display:none;隐藏元素，渲染树不会包含该渲染对象，因此该元素不会在页面中占据位置，也不会响应绑定的监听事件。
-（2）使用 visibility:hidden;隐藏元素。元素在页面中仍占据空间，但是不会响应绑定的监听事件。
-（3）使用 opacity:0;将元素的透明度设置为 0，以此来实现元素的隐藏。元素在页面中仍然占据空间，并且能够响应元素绑定的监听事件。
-（4）通过使用绝对定位将元素移除可视区域内，以此来实现元素的隐藏。
-（5）通过 z-index 负值，来使其他元素遮盖住该元素，以此来实现隐藏。
-（6）通过 clip/clip-path 元素裁剪的方法来实现元素的隐藏，这种方法下，元素仍在页面中占据位置，但是不会响应绑定的监听事件。
-（7）通过 transform:scale(0,0)来将元素缩放为 0，以此来实现元素的隐藏。这种方法下，元素仍在页面中占据位置，但是不会响应绑定的监听事件。
详细资料可以参考： 《CSS 隐藏元素的八种方法》

2.99.css 实现上下固定中间自适应布局？

利用绝对定位实现 body {
  padding: 0;
  margin: 0;
}

.header {
  position: absolute;
  top: 0;
  width: 100%;
  height: 100px;
  background: red;
}

.container {
  position: absolute;
  top: 100px;
  bottom: 100px;
  width: 100%;
  background: green;
}

.footer {
  position: absolute;
  bottom: 0;
  height: 100px;
  width: 100%;
  background: red;
}

利用 flex 布局实现 html,
body {
  height: 100%;
}

body {
  display: flex;
  padding: 0;
  margin: 0;
  flex-direction: column;
}

```
.header {
    height: 100px;
    background: red;
}

.container {
    flex-grow: 1;
    background: green;
}

.footer {
    height: 100px;
    background: red;
}
```

详细资料可以参考： 《css 实现上下固定中间自适应布局》

2.100.css 两栏布局的实现？

相关资料：

```
/*两栏布局一般指的是页面中一共两栏，左边固定，右边自适应的布局，一共有四种实现的方式。*//*以左边宽度固定为 200px 为例*/
/*（1）利用浮动，将左边元素宽度设置为 200px，并且设置向左浮动。将右边元素的 margin-left 设置为 200px，宽度设置为 auto（默认为 auto，撑满整个父元素）。*/
.outer {
    height: 100px;
}

.left {
    float: left;

    height: 100px;
    width: 200px;

    background: tomato;
}

.right {
    margin-left: 200px;

    width: auto;
    height: 100px;

    background: gold;
}
```

```
/*（2）第二种是利用 flex 布局，将左边元素的放大和缩小比例设置为 0，基础大小设置为 200px。将右边的元素的放大比例设置为 1，缩小比例设置为 1，基础大小设置为 auto。*/
.outer {
    display: flex;

    height: 100px;
}

.left {
    flex-shrink: 0;
    flex-grow: 0;
    flex-basis: 200px;

    background: tomato;
}

.right {
    flex: auto;
    /*11auto*/

    background: gold;
}
/*（3）第三种是利用绝对定位布局的方式，将父级元素设置相对定位。左边元素设置为 absolute 定位，并且宽度设置为 200px。将右边元素的 margin-left 的值设置为 200px。*/
.outer {
    position: relative;

    height: 100px;
}

.left {
    position: absolute;

    width: 200px;
    height: 100px;

    background: tomato;
}

.right {
    margin-left: 200px;
    height: 100px;

    background: gold;
```

```
}
/*（4）第四种还是利用绝对定位的方式，将父级元素设置为相对定位。左边元素宽度设置为 200px，右边元素设置为绝对定位，左边定位为 200px，其余方向定位为 0。*/
.outer {
    position: relative;

    height: 100px;
}

.left {
    width: 200px;
    height: 100px;

    background: tomato;
}

.right {
    position: absolute;

    top: 0;
    right: 0;
    bottom: 0;
    left: 200px;

    background: gold;
}
```

《两栏布局 demo 展示》

回答：

两栏布局一般指的是页面中一共两栏，左边固定，右边自适应的布局，一共有四种实现的方式。

以左边宽度固定为 200px 为例

-（1）利用浮动，将左边元素宽度设置为 200px，并且设置向左浮动。将右边元素的 margin-left 设置为 200px，宽度设置为 auto（默认为 auto，撑满整个父元素）。

-（2）第二种是利用 flex 布局，将左边元素的放大和缩小比例设置为 0，基础大小设置为 200px。将右边的元素的放大比例设置为 1，缩小比例设置为 1，基础大小设置为 auto。

-（3）第三种是利用绝对定位布局的方式，将父级元素设置相对定位。左边元素设置为 absolute 定位，并且宽度设置为 200px。将右边元素的 margin-left 的值设置为 200px。

-（4）第四种还是利用绝对定位的方式，将父级元素设置为相对定位。左边元素宽度设置为 200px，右边元素设置为绝对定位，左边定位为 200px，其余方向定位为 0。

2.101.css 三栏布局的实现？

相关资料：

```
/*三栏布局一般指的是页面中一共有三栏，左右两栏宽度固定，中间自适应的布局，一共有五种实现方式。这里以左边宽度固定为 100px，右边宽度固定为 200px 为例。*/
/*（1）利用绝对定位的方式，左右两栏设置为绝对定位，中间设置对应方向大小的 margin
```

```
的值。*/
.outer {
    position: relative;
    height: 100px;
}

.left {
    position: absolute;

    width: 100px;
    height: 100px;
    background: tomato;
}

.right {
    position: absolute;
    top: 0;
    right: 0;

    width: 200px;
    height: 100px;
    background: gold;
}

.center {
    margin-left: 100px;
    margin-right: 200px;
    height: 100px;
    background: lightgreen;
}
/*（2）利用 flex 布局的方式，左右两栏的放大和缩小比例都设置为 0，基础大小设置为固定的大小，中间一栏设置为 auto*/
.outer {
    display: flex;
    height: 100px;
}

.left {
    flex: 00100px;
    background: tomato;
}

.right {
    flex: 00200px;
```

```
    background: gold;
}

.center {
    flex: auto;
    background: lightgreen;
}
/*（3）利用浮动的方式，左右两栏设置固定大小，并设置对应方向的浮动。中间一栏设置左右两个方向的 margin 值，注意这种方式，中间一栏必须放到最后。*/
.outer {
    height: 100px;
}

.left {
    float: left;
    width: 100px;
    height: 100px;
    background: tomato;
}

.right {
    float: right;
    width: 200px;
    height: 100px;
    background: gold;
}

.center {
    height: 100px;
    margin-left: 100px;
    margin-right: 200px;
    background: lightgreen;
}
/*（4）圣杯布局，利用浮动和负边距来实现。父级元素设置左右的 padding，三列均设置向左浮动，中间一列放在最前面，宽度设置为父级元素的宽度，因此后面两列都被挤到了下一行，通过设置 margin 负值将其移动到上一行，再利用相对定位，定位到两边。*/
.outer {
    height: 100px;
    padding-left: 100px;
    padding-right: 200px;
}

.left {
    position: relative;
```

```
    left: -100px;

    float: left;
    margin-left: -100%;

    width: 100px;
    height: 100px;
    background: tomato;
}

.right {
    position: relative;
    left: 200px;

    float: right;
    margin-left: -200px;

    width: 200px;
    height: 100px;
    background: gold;
}

.center {
    float: left;

    width: 100%;
    height: 100px;
    background: lightgreen;
}
/*（5）双飞翼布局，双飞翼布局相对于圣杯布局来说，左右位置的保留是通过中间列的 margin 值来实现的，而不是通过父元素的 padding 来实现的。本质上来说，也是通过浮动和外边距负值来实现的。*/

.outer {
    height: 100px;
}

.left {
    float: left;
    margin-left: -100%;

    width: 100px;
    height: 100px;
    background: tomato;
```

```
}

.right {
    float: left;
    margin-left: -200px;

    width: 200px;
    height: 100px;
    background: gold;
}

.wrapper {
    float: left;

    width: 100%;
    height: 100px;
    background: lightgreen;
}

.center {
    margin-left: 100px;
    margin-right: 200px;
    height: 100px;
}
```

《三栏布局 demo 展示》

回答：

三栏布局一般指的是页面中一共有三栏，左右两栏宽度固定，中间自适应的布局，一共有五种实现方式。这里以左边宽度固定为 100px，右边宽度固定为 200px 为例。

（1）利用绝对定位的方式，左右两栏设置为绝对定位，中间设置对应方向大小的 margin 的值。

（2）利用 flex 布局的方式，左右两栏的放大和缩小比例都设置为 0，基础大小设置为固定的大小，中间一栏设置为 auto。

（3）利用浮动的方式，左右两栏设置固定大小，并设置对应方向的浮动。中间一栏设置左右两个方向的 margin 值，注意这种方式，中间一栏必须放到最后。

（4）圣杯布局，利用浮动和负边距来实现。父级元素设置左右的 padding，三列均设置向左浮动，中间一列放在最前面，宽度设置为父级元素的宽度，因此后面两列都被挤到了下一行，通过设置 margin 负值将其移动到上一行，再利用相对定位，定位到两边。双飞翼布局中间列的宽度不能小于两边任意列的宽度，而双飞翼布局则不存在这个问题。

（5）双飞翼布局，双飞翼布局相对于圣杯布局来说，左右位置的保留是通过中间列的 margin 值来实现的，而不是通过父元素的 padding 来实现的。本质上来说，也是通过浮动和外边距负值来实现的。

2.102.实现一个宽高自适应的正方形

```
/*1.第一种方式是利用 vw 来实现*/
.square {
  width: 10%;
  height: 10vw;
  background: tomato;
}
/*2.第二种方式是利用元素的 margin/padding 百分比是相对父元素 width 的性质来实现*/
.square {
  width: 20%;
  height: 0;
  padding-top: 20%;
  background: orange;
}
/*3.第三种方式是利用子元素的 margin-top 的值来实现的*/
.square {
  width: 30%;
  overflow: hidden;
  background: yellow;
}

.square::after {
  content: "";
  display: block;
  margin-top: 100%;
}
```

《自适应正方形 demo 展示》

2.103.实现一个三角形

```
/*三角形的实现原理是利用了元素边框连接处的等分原理。*/
.triangle {
  width: 0;
  height: 0;
  border-width: 100px;
  border-style: solid;
  border-color: tomatotransparenttransparenttransparent;
}
```

《三角形 demo 展示》

2.104.一个自适应矩形，水平垂直居中，且宽高比为 2:1

```
/*实现原理参考自适应正方形和水平居中方式*/
.box {
    position: absolute;
    top: 0;
    right: 0;
    left: 0;
    bottom: 0;
    margin: auto;

    width: 10%;
    height: 0;
    padding-top: 20%;
    background: tomato;
}
```