## 讲一下你对盒模型的理解？

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

盒模型有两种：IE盒模型和标准盒模型。两者的区别是标准和模型的宽高不包含border和padding，而IE盒模型的宽高包含border和padding。

使用box-sizing属性可以切换和模型，默认值为content-box，即标准盒模型，border-box则是IE盒模型。

## css 选择器优先级？

!important > 行内样式（比重1000）> ID 选择器（比重100） > 类选择器（比重10） > 标签（比重1） > 通配符 > 继承 > 浏览器默认属性

## 说一说样式优先级的规则是什么？

得分点： !important、行内样式、嵌入样式、外链样式、id选择器、类选择器、标签选择器、复合选择器、通配符、继承样式

标准回答

CSS样式的优先级应该分成五大类

第一类!important，无论引入方式是什么，选择器是什么，它的优先级都是最高的。

第二类引入方式，行内样式的优先级要高于嵌入和外链，嵌入和外链如果使用的选择器相同就看他们在页面中插入的顺序，在后面插入的会覆盖前面的。

第三类选择器，选择器优先级：id选择器>（类选择器 | 伪类选择器 | 属性选择器 ）> （后代选择器 | 伪元素选择器 ）> （子选择器 | 相邻选择器） > 通配符选择器 。

第四类继承样式，是所有样式中优先级比较低的。

第五类浏览器默认样式优先级最低。

## 垂直居中几种方式？

单行文本: line-height = height

图片: vertical-align: middle;

absolute 定位: top: 50%;left: 50%;transform: translate(-50%, -50%);

flex: display:flex;margin:auto

## 简明说一下 CSS link 与 @import 的区别和用法？

link 是 XHTML 标签，除了加载CSS外，还可以定义 RSS 等其他事务；@import 属于 CSS 范畴，只能加载 CSS。

link 引用 CSS 时，在页面载入时同时加载；@import 需要页面网页完全载入以后加载。

link 是 XHTML 标签，无兼容问题；@import 是在 CSS2.1 提出的，低版本的浏览器不支持。

link 支持使用 Javascript 控制 DOM 去改变样式；而@import不支持。

## rgba和opacity的透明效果有什么不同？

opacity 会继承父元素的 opacity 属性，而 RGBA 设置的元素的后代元素不会继承不透明属性。

## display:none和visibility:hidden的区别？

display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。  
visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

## position的值， relative和absolute分别是相对于谁进行定位的？

relative:相对定位，相对于自己本身在正常文档流中的位置进行定位。

absolute:生成绝对定位，相对于最近一级定位不为static的父元素进行定位。

fixed: （老版本IE不支持）生成绝对定位，相对于浏览器窗口或者frame进行定位。

static:默认值，没有定位，元素出现在正常的文档流中。

sticky:生成粘性定位的元素，容器的位置根据正常文档流计算得出。

## 画一条0.5px的直线？

考查的是css3的transform

height: 1px;

transform: scale(0.5);

## calc, support, media各自的含义及用法？

1. @support 主要是用于检测浏览器是否支持CSS的某个属性，其实就是条件判断，如果支持某个属性，你可以写一套样式，如果不支持某个属性，你也可以提供另外一套样式作为替补。
2. calc() 函数用于动态计算长度值。 calc()函数支持 “+”, “-”, “\*”, “/” 运算；
3. @media 查询，你可以针对不同的媒体类型定义不同的样式。

## 1rem、1em、1vh、1px各自代表的含义？

rem

rem是全部的长度都相对于根元素元素。通常做法是给html元素设置一个字体大小，然后其他元素的长度单位就为rem。

em

子元素字体大小的em是相对于父元素字体大小

元素的width/height/padding/margin用em的话是相对于该元素的font-size

vw/vh

全称是 Viewport Width 和 Viewport Height，视窗的宽度和高度，相当于 屏幕宽度和高度的 1%，不过，处理宽度的时候%单位更合适，处理高度的 话 vh 单位更好。

px

px像素（Pixel）。相对长度单位。像素px是相对于显示器屏幕分辨率而言的。

一般电脑的分辨率有{19201024}等不同的分辨率

19201024 前者是屏幕宽度总共有1920个像素,后者则是高度为1024个像素

## 画一个三角形？

.a {

width: 0;

height: 0;

border-width: 100px;

border-style: solid;

border-color: transparent #0099CC transparent transparent;

transform: rotate(90deg); /\*顺时针旋转90°\*/

}

<div class="a"></div>

## BFC 是什么？

BFC 即 Block Formatting Contexts (块级格式化上下文)，它属于普通流，即：元素按照其在 HTML 中的先后位置至上而下布局，在这个过程中，行内元素水平排列，直到当行被占满然后换行，块级元素则会被渲染为完整的一个新行，除非另外指定，否则所有元素默认都是普通流定位，也可以说，普通流中元素的位置由该元素在 HTML 文档中的位置决定。

可以把 BFC 理解为一个封闭的大箱子，箱子内部的元素无论如何翻江倒海，都不会影响到外部。

只要元素满足下面任一条件即可触发 BFC 特性

* body 根元素
* 浮动元素：float 除 none 以外的值
* 绝对定位元素：position (absolute、fixed)
* display 为 inline-block、table-cells、flex
* overflow 除了 visible 以外的值 (hidden、auto、scroll)

## 常见兼容性问题？

* 浏览器默认的margin和padding不同。解决方案是加一个全局的\*{margin:0;padding:0;}来统一。
* Chrome 中文界面下默认会将小于 12px 的文本强制按照 12px 显示,

可通过加入 CSS 属性 -webkit-text-size-adjust: none; 解决.

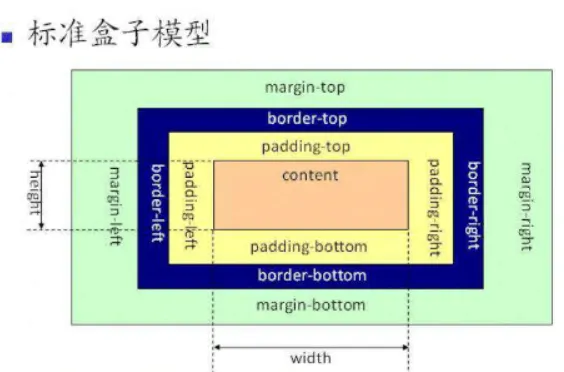
## 块级元素、行内元素分别有哪些？

块级元素：div , p , form, ul, li , ol, dl, form, address, fieldset, hr, menu, table 行内元素：span, strong, em, br, img , input, label, select, textarea, cite

## 盒模型的差异？

css盒模型本质是一个盒子，它由边距、边框、填充和实际内容组成。盒模型能够让我们在其他元素和周边元素边框之间的空间放置元素。

标准盒与怪异盒的区别在于他们的总宽度的计算公式不一样。标准模式下总宽度=width+margin（左右）+padding（左右）border（左右）；怪异模式下总宽度=width+margin（左右）（就是说width已经包含了padding和border值）。标准模式下如果定义的DOCTYPE缺失，则在ie6、ie7、ie8下汇触发怪异模式。当设置为box-sizing:content-box时，将采用标准模式解析计算，也是默认模式；当设置为box-sizing:border-box时，将采用怪异模式解析计算。



## 说一说你是怎么理解BFC的？

得分点： 块级格式化上下文、独立的渲染区域、不会影响边界以外的元素、形成BFC条件、float、position、overflow、display

标准回答： BFC(Block Formatting Context)块级格式化上下文，是Web页面一块独立的渲染区域，内部元素的渲染不会影响边界以外的元素。

BFC布局规则：

内部盒子会在垂直方向，一个接一个地放置。

Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠。

每个盒子（块盒与行盒）的margin box的左边，与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。

BFC的区域不会与float box重叠。

BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。

计算BFC的高度时，浮动元素也参与计算。

BFC形成的条件:

根元素：body；

元素设置浮动：float 除 none 以外的值；

元素设置绝对定位：position(absolute、fixed)；

display值为：inline-block、table-cell、table-caption、flex 等；

overflow值为：hidden、auto、scroll；

BFC解决能的问题：

解决 margin 的重叠问题：由于 BFC 是一个独立的区域，内部的元素和外部的元素互不影响，将两个元素变为两个 BFC，就解决了 margin 重叠的问题。

解决高度塌陷的问题：在对子元素设置浮动后，父元素会发生高度塌陷，也就是父元素的高度变为 0。解决这个问题，只需要把父元素变成一个BFC。常用的办法是给父元素设置overflow:hidden。

创建自适应两栏布局：可以用来创建自适应两栏布局：左边的宽度固定，右边的宽度自适应。

## 清除浮动有哪些方案？

清除浮动主要是为了解决父元素因为子元素浮动引起高度塌陷的问题。

本质上清除浮动的方式有两种，一种是利用clear属性，一种是利用BFC。

clear属性清除浮动又包括空标签法和伪元素法两种，代码如下：

使用空标签清除浮动 在浮动元素的末尾添加一个空标签，并设置 clear: both 属性，即可清除浮动。例如：

<div class="float-parent">

<div class="float-left"></div>

<div class="float-right"></div>

<div class="clearfix"></div>

</div>

.clearfix {

clear: both;

}

使用 :after 伪元素清除浮动

<div class="float-parent">

<div class="float-left"></div>

<div class="float-right"></div>

<div class="clearfix"></div>

</div>

.float-parent:after {

content: "";

display: table;

clear: both;

}

BFC清除浮动的方式有多种，我这里就以overflow为例：

<div class="float-parent" style="overflow: hidden;">

<div class="float-left"></div>

<div class="float-right"></div>

</div>

## 伪类和伪元素有什么区别？

伪类表示被选择元素的某种状态或者选择是，常见的伪类有三种，状态类、结构类和表单类。例如:hover就是状态类伪类，:first-child就是结构类伪类，:cheked就是表单类伪类。

a:hover { color: red; } /\* 鼠标悬停在链接上时文字变成红色 \*/

input:checked + label { background-color: green; } /\* 选中表单元素时相邻的标签背景变成绿色 \*/

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。伪元素常见的用途有给元素添加小部件、小图标以及清除浮动。 例如：

.arrow::before {

content: "";

display: block;

width: 0;

height: 0;

border-top: 10px solid transparent;

border-bottom: 10px solid transparent;

border-right: 10px solid red;

}

另外，虽然没有强制规定，但是伪类通常是单冒号，伪元素通常是用双冒号。

## 怎么做移动端适配？

常用的移动端适配方案有以下几种：

**rem方案**：淘宝的移动端适配方案，使用相对单位rem结合JS动态计算rem值来实现移动端适配，将页面在不同尺寸的屏幕小按照宽度等比例缩放。

**vw方案**：和rem方案类似，只是单位换成了vwborder方案：rem和vw方案都是等比例缩放，但是对于一些对UI要求特别高的大厂项目，缩放的显示效果并不是最佳，这时候也可以和UI配合采取px绝对像素单位。

**px方案**：rem和vw方案都是等比例缩放，但是对于一些对UI要求特别高的大厂项目，缩放的显示效果并不是最佳，这时候也可以和UI配合采取px绝对像素单位。

**媒体查询**：对于一些具体的场景，可以根据不同设备的像素区间来针对性地编写样式，这时候就使用媒体查询。

**百分比布局**：将元素的宽度和高度设置为百分比，使得页面可以根据不同的屏幕尺寸进行等比例缩放，这种方案和rem、vw原理类似，但是计算比较困难，而且百分比相对的元素不固定，容易使问题变得复杂。

**响应式布局**：对于一些定制化程度要求不高，但是需要PC和移动两端共用一套代码的场景，可以使用一些响应式布局的样式库，比如Bootstrap和Tailwind。

## 移动端1px边框问题怎么解决？

由于不同的设备屏幕像素密度的不同，一些边框、线条等细节元素的显示可能会出现“1px问题”，即在某些设备上，本应该显示为1像素的边框或者线条，实际上却被放大成了2像素或者更多像素，导致显示效果不佳。

常见的解决方案：

使用border-image：使用border-image可以将图片作为边框来显示，避免了使用CSS边框样式时的1px问题。

使用box-shadow：使用box-shadow代替边框，然后将边框设为透明，可以避免1px问题。

.border {

box-shadow: 0 0 0 1px #ccc;

}

3. border + transform：使用transform将边框缩小一半。

.border {

border: 1px solid #ccc;

transform: scaleY(0.5);

}

4. 伪元素 + transform：与3类似，不同的是用伪元素来实现。

.border:before{

content: "";

display: block;

position: absolute;

left: 0;

top: 0;

bottom: 0;

right: 0;

border: 1px solid #ccc;

transform-origin: 0 0;

transform: scaleY(0.5);

}

在父元素上添加一个伪元素，然后给伪元素设置一个边框，并将其缩小为0.5倍。这样就可以实现1px的边框效果了。

1. 使用viewport单位：使用viewport相关的单位（如vw、vh、vmin和vmax）来设置边框或者线条的大小，可以让元素的大小自适应不同的设备像素密度。

## CSS有哪些主流布局方式？

CSS主要有五大主流布局：

**flex布局**，目前应用最广的布局方式，虽然不兼容IE低版本，但是就连微软自己都已经放弃IE11了。

**浮动布局**，需要清除浮动，副作用较多，现在使用的已经不多了，除了在部分PC端需要兼容IE低版本的项目中。

**定位布局**，主要用来实现定位效果，依然使用频繁。

**栅格布局**，面向未来的布局，只是现在兼容性还不是很好，且学习成本比较高。

**table布局**，比浮动布局还要老的布局，现在已经很少使用，不过兼容性特别好。

可以根据自己的日常开发经验以及这五种布局方式的优缺点和适用场景谈一谈对它们的理解。

## 对CSS动画有了解吗，CSS实现动画有哪几种方式？

CSS动画主要包括transition动画、transform动画和animation动画。

1. **transition动画**：主要用来实现过渡效果，包括颜色、透明度、大小、位置，从一个状态向另一个状态过渡。

.element {

transition: property duration timing-function delay;

}

1. **transform动画**：用于对元素进行变形和转换，如旋转、缩放、平移和倾斜等。transform可以配合transition一起使用，例如：

.element {

transform: rotate(45deg);

transition: transform duration timing-function delay;

}

1. **animation动画**：使用@keyframes规则定义一组关键帧，描述元素在动画过程中的样式变化。然后通过animation属性将关键帧应用到元素上：

@keyframes animationName {

0% { /\* 初始样式 \*/ }

50% { /\* 中间样式 \*/ }

100% { /\* 结束样式 \*/ }

}

.element {

animation: animationName duration timing-function delay iteration-count direction;

}

在日常开发中大多数时候制作简单动画使用一些第三方的CSS动画库就足够了，比如Animate.css、Hover.css，但是复杂动画不仅要考虑CSS，还要考虑JS。在Vue和React这样的框架中，都提供了Transition组件来让我们实现动画。

对于复杂动画，除了实现效果之外还要考虑到动画的性能优化，需要使用Chrome工具去调优。

## 什么是CSS工程化？

这是一个偏主观性的题，可以根据自己的项目经历从以下几个方面来回答：

预编译器：less、sass、stylus等。

CSS模块化方案：常见的像Vue里的scoped方案以及React里的CSS-in-JS方案。

CSS自动化工具：比如PostCSS，就提供很多有用的插件，让我们可以做一些CSS的自动化工作，比如自动添加前缀、自动格式化代码、移动端适配。

代码检测工具：如CSSLint、Stylelint等，可以自动化检测CSS代码的错误和潜在问题，提高代码质量和稳定性。

样式指南：主要是指CSS的规范，包括设计原则、排版规则、颜色使用、图标规范、命名规范、代码格式化等各方面的规范。

## CSS有哪些优化手段？

CSS优化，可以从以下方面来考虑：

压缩CSS代码：通过删除空格、注释、冗余代码等方法，可以减小CSS文件的大小，提高加载速度，不过这些通常脚手架已经帮我们做好了。

合并CSS文件：将多个CSS文件合并成一个文件，减少HTTP请求次数，提高加载速度。在项目中可以用Webpack插件来实现。

首屏样式内联，可以减少加载CSS文件的时间，提升页面加载速度。

使用CSS3硬件加速，可以减少回流重绘，常见的触发硬件加速的CSS属性有：transform、opacity、filters、Will-change。

减少使用通配符和后代选择器：过多的通配符和后代选择器会降低选择器的匹配速度，应尽量减少使用。

避免使用过多的float：过多的浮动会导致页面回流和重绘，影响性能。

缓存：可以在服务器端设置协商缓存或者强缓存来缓存CSS请求。

日常开发中，除了开发CSS动画之外，绝大多数情况下，CSS的性能不会出现太明显的问题。所以在绝大多数情况下，只要是按照基本的规则和规范写CSS代码，问题就不大。

## 为什么要做样式初始化？

因为浏览器有默认样式，而且不同浏览器默认样式不一样，为了让样式显示一致，要去掉这些默认样式。

常见的方案有：

reset.css：将所有元素的样式都设置为相同的初始值，以消除不同浏览器之间的差异。这种方式需要注意的是，一些元素的样式可能与开发者所期望的有所不同，因此需要进行特殊处理。

Normalize.css：只重置一部分元素的样式，而不是重置所有元素。这种方式可以避免一些样式上的问题，同时还可以保留一些元素的默认样式，提高代码的可维护性和可读性。

## 讲一下什么是重绘和回流？

重绘（Repaint）：重绘是指当元素的样式改变，但不影响其在文档流中的位置时，浏览器会将新样式直接绘制到屏幕上，这个过程称为重绘。

回流（Reflow）：当元素的尺寸、位置或其他属性发生变化时，浏览器需要重新计算页面中所有元素的位置和大小，并且将它们绘制到屏幕上，这个过程称为回流，又叫做重排。

回流的代价比重绘高得多，因为回流会导致浏览器重新计算页面布局，这个过程中涉及到大量的计算和DOM操作，所以需要尽量减少回流的次数。以下是一些可能导致回流的操作：

页面初始渲染时。

调整窗口大小时。

改变字体大小。

改变元素的大小或位置。

修改元素的内容，例如输入框中的文字。

可以通过以下方式来减少回流的次数：

使用transform替代top或left。

使用visibility替换display: none，因为前者只会引起重绘，后者会引发回流（改变了布局）。

不要把DOM节点的属性值放在一个循环里当成循环里的变量。

尽量不要使用table布局，可能很小的一个小改动就会造成整个table的重新布局。

尽量减少DOM的嵌套层级，减少不必要的父子元素节点。