43 重点布局方案（上）

更新时间：2020-06-11 09:48:11

> 今天应做的事没有做，明天再早也是耽误了。——裴斯泰洛齐

时下很少会有公司开设专门的“CSS 工程师”岗位，前端工程师的竞争力主要靠 JS 及其周边生态的相关知识点来凸显。因此 CSS 相关的考题其实悬念不多，大家精做好题即可。

针对 CSS，我们从两个角度来做面试准备。

首先是基本布局方案，这个是重中之重；其次是响应式的布局方案，这个里面有一些概念是需要大家特别了解的，以防到时候被面试官整懵。这两个专题（尤其是当涉及到实战代码部分时），在实际面试中有着非常高的区分度。这里我们直接上手面试中最不容易做好的一部分高频考题，做熟做透就不虚。

深挖面试官都喜欢的垂直/水平居中问题

将某个元素在容器中水平居中，你有几种思路？这是一道大题，如果你较起真来，实际上能刨出非常多的方法。我之前看到有人写了一篇很长的文章，里面仔仔细细竟然数出十几种方案。

诚然，像这样的问题，答案肯定是多多益善的。不过从实际的角度出发，真要背十几种方案实在不现实。真实面试情境下，面试官也不会有耐心听候选人像数流水账一样去数太多出来。就这道题来说，**重要的不是你掌握的方法数量，而是你对已经掌握的方法到底理解到什么程度**。因此我在这里给大家着重讲的就是三个方向的思路，同时在讲解过程中，我会给大家穿插一些 CSS 基础、CSS 进阶程度的理论知识剖析。

注：这个题就算你非常自信自己会做，也不要跳过这节的讲解。**很多同学只是会写代码，但是对相关的原理却说不清楚，在面试时会吃很大的亏**。

思路一：绝对定位方案

基于绝对定位来做这个题，我们有三条路可以走：

1. 常规操作：margin 置为负值

这条路的前提是，div 盒子的宽高提前已知。

首先咱们初始化一个容器和一个 div ：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS垂直居中测试</title>
  <style>
    #center {
      width: 200px;
      height: 200px;
      background-color: red;
    }

    html, body, #container {
      width: 100%;
      height: 100%;
      background-color: black;
    }
  </style>
</head>

<body>
  <div id="container">
    <div id="center"></div>
  </div>
</body>

</html>
```

现在在这个容器里， id 为 center 的 div 还没有进行任何的居中处理：

现在我们给 center 一个绝对定位，让它距离左侧边缘和顶部边缘分别 50%：

```
#center {
  width: 200px;
  height: 200px;
  background-color: red;
  position: absolute;
  left: 50%;
  top: 50%;
}
```

因为我们是相对父元素定位，这里别忘了给父元素设置一个 relative：

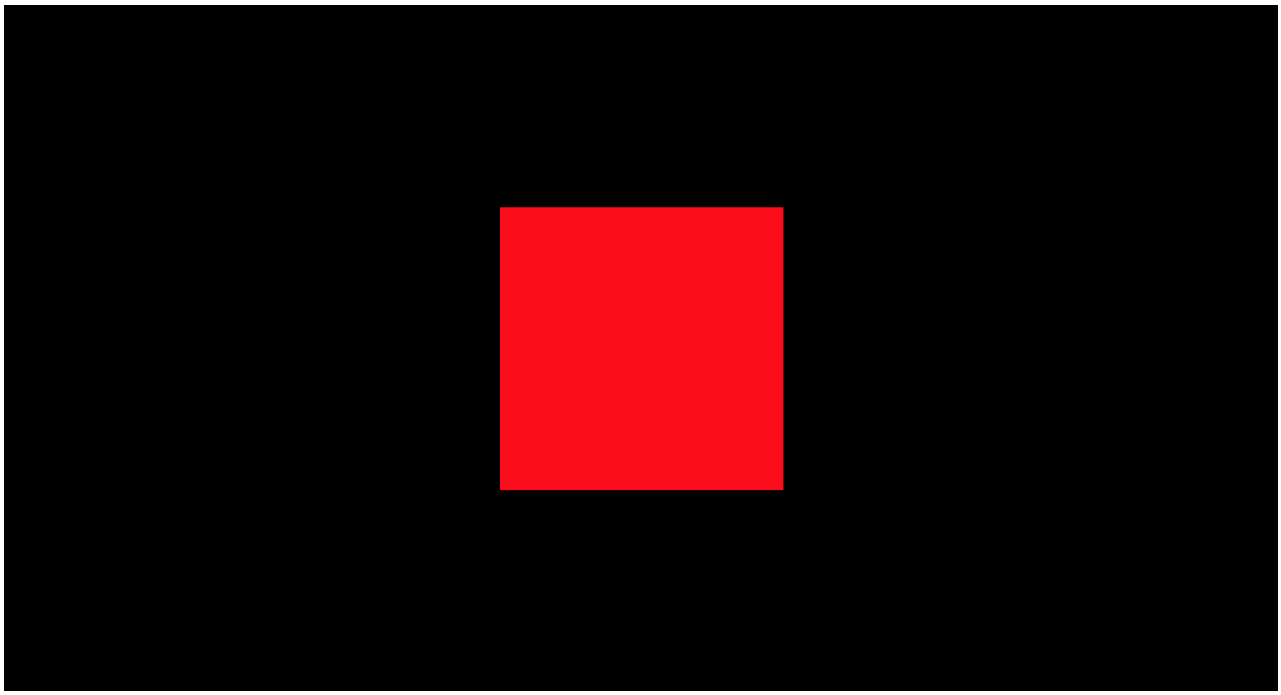
```
#container {
  position: relative;
}
```

绝对定位处理完后效果如下：

大家会发现直接绝对定位保持上下左右各 50% 的距离，并不能达到我们的目的。绝对定位的基准线是元素的左侧边缘和顶部边缘，而我们的居中所期望的却是元素的中心和容器边缘保持上下左右各 50% 的距离。因此我们就要对元素边缘和元素中心的这个距离进行处理。在这个场景下，元素的宽高都是已知的，因此我们直接设置负的 margin 值即可：

```
#center {
  width: 200px;
  height: 200px;
  background-color: red;
  position: absolute;
  left: 50%;
  top: 50%;
  margin-left: -100px;
  margin-top: -100px;
}
```

margin-left 设为负值，可以让元素相对于自己原有的位置向左移动相应的距离，margin-top 同理。通过我们这样的操作，元素就会相对于它原来的位置分别向左和向上移动自身宽/高的一半距离。如此，我们就可以弥补元素中心到元素左侧边缘和元素顶部边缘的差距了：



这样一来，垂直/水平居中就大功告成了。

2. 深入理解元素的流体特性：神奇的 margin: auto

实际的开发中，当我们想要水平居中某个元素时，可以把它的 margin 设为如下值：

```
margin: 0 auto;
```

为什么设置 auto 可以让元素相对于父元素水平居中？这点很多候选人都答不上来。其实，这和 auto 这个属性的取值有关。auto 在任何情况下，只会取下面两种值中的一个：

1. 父元素剩余空间的宽度
2. 0

何时取1，何时取2？

这是由元素的布局方式决定的，当元素的布局方式为 static/relative 且宽高已知时，auto 取1中的值；当元素的布局方式为 postion/absolute/fixed 或者 float/inline 或者宽高未知时，auto 就取 2中的值。

注意，以上 auto 的取值均指水平方向，垂直方向上，auto 不会自动填充。

这就引出了我们接下来要解决的问题：如何利用 auto 实现元素的垂直居中？

答案是利用元素的**流体特性**。所谓流体特性，看上去很高级，实际非常简单：当一个绝对定位元素，其对立定位方向属性同时有具体定位数值的时候，流体特性就发生了。

流体特性的妙处，在于**元素可自动填充父级元素的可用尺寸**。

当流体特性发生时，我们可以给水平/垂直方向的对立定位（也就是 left、right、top、bottom）各设定一个值，然后将水平/垂直方向的 margin 均设为 auto，这样一来，auto 就会自动平分父元素的剩余空间了。

基于这个理解，我们可以这样来写 center 的 css 代码：

```
#center {
  background-color: red;
  width: 200px;
  height: 200px;
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  margin: auto;
}
```

注意，这里的 left、right、top、bottom 的值其实没有实际的大小描述意义。我们只要确认其存在性就行，就算不是 0，也可以激发其流体特性：

```
#center {
  background-color: red;
  width: 200px;
  height: 200px;
  position: absolute;
  top: 10px;
  bottom: 10px;
  left: 20px;
  right: 20px;
  margin: auto;
}
```

3. 动画属性来帮忙

可惜我们并不是在所有的场景下都已知子元素的宽高。在宽高不定的情况下，我们一般是使用 transform 这个属性来达到目的：

```
#center {
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
}
```

transform 是 css3 引入的一个动画属性，它允许我们对元素进行旋转、缩放、移动或倾斜。这里我们用到的 translate，对应的就是它的移动能力。

translate 接受两个参数，分别对应元素沿 X 轴的移动量和沿 Y 轴的移动量。这里我们两个都填了 50%，意思就是元素需要横向/纵向分别移动自身宽度/高度的 50%。

完整的示例如下，大家可以丢进自己的浏览器里观摩一下效果：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS垂直居中测试</title>
  <style>
    #center {
      background-color: red;
      position: absolute;
      left: 50%;
      top: 50%;
      transform: translate(-50%, -50%);
    }

    html, body, #container {
      width: 100%;
      height: 100%;
      background-color: black;
    }
    #container {
      position: relative;
    }
  </style>
</head>

<body>
  <div id="container">
    <div id="center">哈哈哈哈</div>
  </div>
</body>

</html>
```

注：position 布局下，我们还可以借助 CSS 的 calc() 函数来完成同样的调整。这个思路在面试中考察占比不算高，大家感兴趣可以了解下。在作答时间有限的情况下，我们的目标就是答好本文所展开讲解的这几个点。

思路二：Flex 布局

利用 flex 布局，事情就简单多了。不管元素是否定高定宽，Flex 布局都能够帮我们轻松做到。

Flex 其实是 Flexible Box 的缩写，翻译过来是"弹性布局"，用来为盒状模型提供最大的灵活性。

关于 Flex 布局的基础知识，这里推荐大家阅读 阮一峰老师的Flex教学。本文仅对 Flex 在居中层面的应用作讲解。

仍然是我们 container + center 的这个demo，我现在把它恢复到最原始的状态：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS垂直居中测试</title>
  <style>
    #center {
      background-color: red;
    }

    html, body, #container {
      width: 100%;
      height: 100%;
      background-color: black;
    }
  </style>
</head>

<body>
  <div id="container">
    <div id="center">哈哈哈哈</div>
  </div>
</body>

</html>
```

首先我们要做的是对父元素 container 应用 Flex 布局：

```
#container {
  display: flex;
}
```

然后指定父元素中子元素的排列方式：这里我们没有对 Flex 容器的主轴和副轴进行特别的设置，因此它的主轴就是水平方向，副轴就是垂直方向。设置主轴的元素排列，我们用 justify-content 属性；设置副轴的元素排列，我们用 align-items 属性，这里我们全部设为 center（居中排列）：

```
#container {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

就可以轻松实现水平/垂直居中了。

思路三：table 布局

最后一个要推荐给大家的布局方式，是历史悠久的 table 布局。虽然现在很多团队现在都已经不用 table 布局了，但是奈何 table 布局兼容性实在是不错，一部分团队仍然对它表示难舍难分，因此 table 布局也是大家答题中有必要答出来的一种解法。

我们仍然是把 DEMO 重置为完全没有进行任何居中处理的状态：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS垂直居中测试</title>
  <style>
    #center {
      background-color: red;
    }

    html, body, #container {
      width: 100%;
      height: 100%;
      background-color: black;
    }
  </style>
</head>

<body>
  <div id="container">
    <div id="center">哈哈哈哈</div>
  </div>
</body>

</html>
```

首先，我们将容器元素的布局方式设置为 table-cell：

```
#container {
  display: table-cell;
}
```

然后指定容器元素内部子元素的布局方式，vertical-align 指定垂直居中，text-align 指定水平居中：

```
#container {
  display: table;
  text-align: center;
  vertical-align: middle;
}
```

然后我们将子元素的布局设为 inline-block，因为 table-cell 理论上只能处理具备行内特性的元素布局：

```
#center {
  background-color: red;
  display: inline-block;
}
```

大家可以把完整的代码丢进浏览器跑跑看：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS垂直居中测试</title>
  <style>
    #center {
      background-color: red;
      display: inline-block;
    }

    #container {
      width: 400px;
      height: 400px;
      background-color: black;
      display: table-cell;
      vertical-align: middle;
      text-align: center;
    }

  </style>
</head>

<body>
  <div id="container">
    <div id="center">哈哈哈哈</div>
  </div>
</body>

</html>
```

注意 table-cell 虽然不要求子元素定高，但是对应父元素的宽高如果继续用百分比的写法会出现问题，这里我们需要给父元素一个确定的宽高值。

移动端疑难杂症：1px 问题如何解决？

1px 问题在实际面试，尤其是大厂面试中出现的频率是比较高的。与此相应的是很多同学简历上虽然写了移动端的开发经验，但实际面试时被问到“1px 问题怎么解决”时却压根不知道面试官在说啥，这样难免会让面试官质疑你对移动端开发这个命题的深入程度。如果你的简历上也写明了移动端/H5页面的开发经历，那么一方面，对于 1px 问题，请一定要引起重视，要言之有物；另一方面，仍然是不必钻牛角尖——1px 问题的解法和垂直/居中布局一样，属于数不胜数系列，我们要做的是抓关键思路，而不是记流水账。

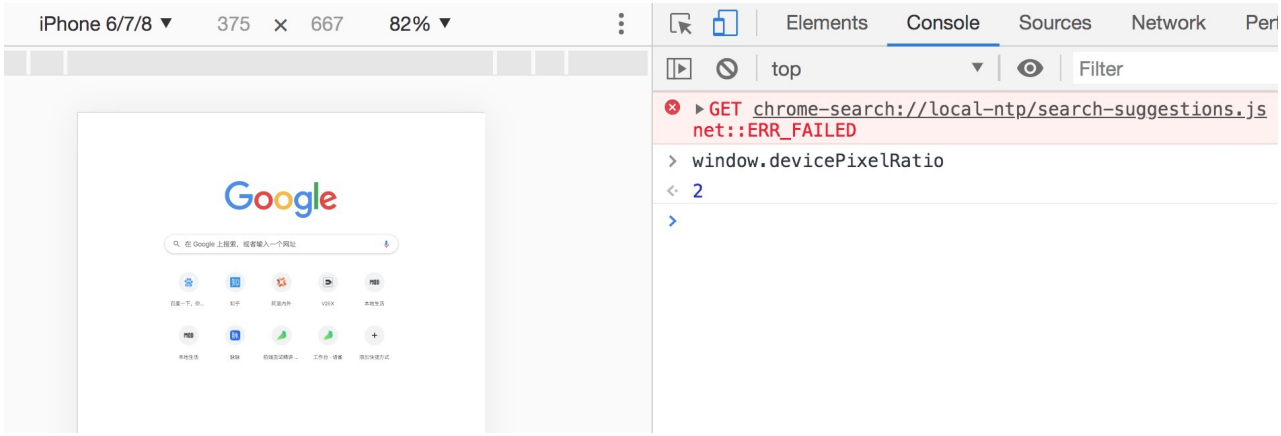
1px 问题的起因

1px 问题指的是：在一些 Retina屏幕 的机型上，移动端页面的 1px 会变得很粗，呈现出不止 1px 的效果。

原因很简单——CSS 中的 1px 并不能和移动设备上的 1px 划等号。它们之间的比例关系有一个专门的属性来描述：

```
window.devicePixelRatio = 设备的物理像素 / CSS像素。
```

大家可以尝试打开自己的 Chrome 浏览器，启动移动端调试模式，然后尝试在控制台去输出这个 devicePixelRatio 的值。这里我选中了 iPhone6/7/8 这系列的机型，输出的结果就是2：



这就意味着我设置的 1px CSS 像素，在这个设备上实际会用 2 个物理像素单元来进行渲染，所以实际看到的一定会比 1px 粗一些。

1px 问题的解决方案

前面咱们说过，1px 问题的解决方案是非常多的。像这种一题多解的情况，一些同学会要求自己记下所有解法，这种精神难能可贵。不过从实用的角度出发，我建议大家掌握3-4种就可以了，记更多解法也是好的，但没有必要。在前端面试这个命题下，最重要的是全局观，也就是说候选人首先要保证的是自己没有知识盲点，每一个知识模块都有一定的知识储备。对于某个具体问题特别执着、精益求精，这应该建立在你对全局知识非常自信的基础上。

话说回来，一道题的实际解法有10种，而你答出3种，这一点也不影响你在面试官心目中的形象。拿 1px 这道问题来说，这个题是我写进团队面试题库的。实际面试中，我们团队对这个题的评价标准是：答出1-2种解法，能说清楚每种解法需要注意什么，就 OK 了——面试是考察你是否具备解决问题的能力，而不是比拼记忆力的斗兽场。

这里我为大家介绍的是实际业务中操作性比较强的三种思路：

思路一：直接写 0.5px

我没开玩笑，比如说你之前 1px 的样式这样写：

```
border:1px solid #333
```

你可以先在 JS 中拿到 window.devicePixelRatio 的值，然后把这个值通过 JSX 或者模板语法给到 CSS 的 data 里，达到这样的效果（这里我用 JSX 语法做个示范）：

```
<div id="container" data-device={{window.devicePixelRatio}}></div>
```

然后你就可以在 CSS 中用属性选择器来命中 devicePixelRatio 为某一值的情况，比如说这里我尝试命中 devicePixelRatio 为2的情况：

```
#container[data-device="2"] {
  border:0.5px solid #333
}
```

直接把 1px 改成 1/devicePixelRatio 后的值，这是目前为止最简单的一种方法。这种方法的缺陷在于兼容性不行，IOS 系统需要8及以上的版本，安卓系统则直接不兼容。

思路二：伪元素先放大后缩小

这个方法的可行性会更高，兼容性也更好。唯一的缺点是代码会变多，哈哈。

我们的思路是**先放大、后缩小：在目标元素的后面追加一个 ::after 伪元素，让这个元素布局为 absolute 之后、整个伸展开铺在目标元素上，然后把它的宽和高都设置为目标元素的两倍，border值设为 1px。接着借助 CSS 动画特效中的放缩能力，把整个伪元素缩小为原来的 50%。此时，伪元素的宽高刚好可以和原有的目标元素对齐，而 border 也缩小为了 1px 的二分之一，间接地实现了 0.5px 的效果。**

代码演示如下：

```
#container[data-device="2"] {
    position: relative;
}

#container[data-device="2"]::after{
      position:absolute;
      top: 0;
      left: 0;
      width: 200%;
      height: 200%;
      content:"";
      transform: scale(0.5);
      transform-origin: left top;
      box-sizing: border-box;
      border: 1px solid #333;
    }
}
```

思路三：viewport 缩放来解决

熟悉移动端开发的同学一定会对 viewport 中的 meta 标签有深刻的印象（没印象也没关系，我们第三节会讲），我们这个思路就是对 meta 标签里几个关键属性下手：

```
<meta name="viewport" content="initial-scale=0.5, maximum-scale=0.5, minimum-scale=0.5, user-scalable=no">
```

这里针对像素比为2的页面，把整个页面缩放为了原来的1/2大小。如此一来，本来占用2个物理像素的 1px 样式，现在占用的就是标准的一个物理像素。根据像素比的不同，这个缩放比例可以被计算为不同的值，我们用 js 代码实现如下：

```
const scale = 1 / window.devicePixelRatio;

// 这里 metaEl 指的是 meta 标签对应的 Dom
metaEl.setAttribute('content', `width=device-width,user-scalable=no,initial-scale=${scale},maximum-scale=${scale},minimum-scale=${scale}`);
```

1px 的 bug 就这样轻松搞定了，但这样做的副作用也很大，整个页面被缩放了。这时候我们的 1px 已经被处理成物理像素大小了，这样的大小在手机上显示边框很合适。但仔细想想，一些原本不需要被缩小的内容，比如文字、图片等，也被无差别缩小掉了。

关于这个副作用的解决，其实有一套完整的方案。但是注意，题目要求我们解决的是 1px 问题，到此为止，1px 问题已经被解决掉了。关于后续问题的解决，我们需要更多的前置知识和概念作为辅助，这里先给大家留个悬念，在本章的第三节中，我们会作更深入和细致的探讨。

如果你对 viewport 这个概念感到一头雾水，那么也没有关系。在本章的第三节，我们会对 viewport、rem 及其相关的方案（包括我们现在正在探讨的这个 1px 问题的解决思路）作进一步的探讨和复盘。

```
}
```

42 跨域解决方案

44 重点布局方案（下）