一面 3：CSS-HTML 知识点与高频考题解析

CSS 和 HTML 是网页开发中布局相关的组成部分，涉及的内容比较多和杂乱，本小节重点介绍下常考的知识点。

知识点梳理

- 选择器的权重和优先级
- 盒模型
  - 盒子大小计算
  - margin 的重叠计算
- 浮动 `float`
  - 浮动布局概念
  - 清理浮动
- 定位 `position`
  - 文档流概念
  - 定位分类
  - fixed 定位特点
  - 绝对定位计算方式
- `flex` 布局
- 如何实现居中对齐?
- 理解语义化
- CSS3 动画
- 重绘和回流

选择器的权重和优先级

CSS 选择器有很多，不同的选择器的权重和优先级不一样，对于一个元素，如果存在多个选择器，那么就需要根据权重来计算其优先级。

权重分为四级，分别是：

1. 代表内联样式，如 `style="xxx"`，权值为 1000；
2. 代表 ID 选择器，如 `#content`，权值为 100；
3. 代表类、伪类和属性选择器，如 `.content`、`:hover`、`[attribute]`，权值为 10；
4. 代表元素选择器和伪元素选择器，如 `div`、`p`，权值为 1。

需要注意的是：通用选择器（*）、子选择器（>）和相邻同胞选择器（+）并不在这四个等级中，所以他们的权值都为 0。 权重值大的选择器其优先级也高，相同权重的优先级又遵循后定义覆盖前面定义的情况。

盒模型

什么是“盒子”

初学 CSS 的朋友，一开始学 CSS 基础知识的时候一定学过`padding` `border`和`margin`，即内边距、边框和外边距。它们三者就构成了一个“盒子”。就像我们收到的快递，本来买了一部小小的手机，收到的却是那么大一个盒子。因为手机白色的包装盒和手机机器之间有间隔层（内边距），手机白色盒子有厚度，虽然很薄（边框），盒子和快递箱子之间还有一层泡沫板（外边距）。这就是一个典型的盒子。

如上图，真正的内容就是这些文字，文字外围有 10px 的内边距，5px 的边框，10px 的外边距。看到盒子了吧?

> 题目：盒子模型的宽度如何计算

固定宽度的盒子

```
<div style="padding:10px; border:5px solid blue; margin: 10px;
width:300px;">
    之前看过一篇文章，叫做《浏览器工作原理：新式网络浏览器幕后揭秘》，
    文章言简意赅的介绍的浏览器的工作过程，web前端
</div>
```

如上图，得到网页效果之后，我们可以用截图工具来量一下文字内容的宽度。发现，文字内容的宽度刚好是 300px，也就是我们设置的宽度。

因此，**在盒子模型中，我们设置的宽度都是内容宽度，不是整个盒子的宽度。而整个盒子的宽度是：（内容宽度 + `border` 宽度 + `padding` 宽度 + `margin` 宽度）之和**。这样我们改四个中的其中一个，都会导致盒子宽度的改变。这对我们来说不友好。

没关系，这个东西不友好早就有人发现了，而且已经解决，下文再说。

充满父容器的盒子

默认情况下，`div`是`display:block`，宽度会充满整个父容器。如下图：

```
<div style="padding:10px; border:5px solid blue; margin: 10px;
width:300px;">
    之前看过一篇文章，叫做《浏览器工作原理：新式网络浏览器幕后揭秘》，
    文章言简意赅的介绍的浏览器的工作过程，web前端
    之前看过一篇文章，叫做《浏览器工作原理：新式网络浏览器幕后揭秘》，
    文章言简意赅的介绍的浏览器的工作过程，web前端
</div>
```

但是别忘记，这个 div 是个盒子模型，它的整个宽度包括（内容宽度 + `border`宽度 + `padding`宽度 + `margin`宽度），整个的宽度充满父容器。

问题就在这里。如果父容器宽度不变，我们手动增大`margin`、`border`或`padding`其中一项的宽度值，都会导致内容宽度的减少。极端情况下，如果内容的宽度压缩到不能再压缩了（例如一个字的宽度），那么浏览器会强迫增加父容器的宽度。这可不是我们想要看到的。

包裹内容的盒子

这种情况下比较简单，内容的宽度按照内容计算，盒子的宽度将在内容宽度的基础上再增加（`padding`宽度 + `border`宽度 + `margin`宽度）之和。

```
<div style="padding:10px; border:5px solid blue; margin: 10px;
width:300px;">
    之前看过一篇文章，叫做《浏览器工作原理：新式网络浏览器幕后揭秘》
</div>
```

`box-sizing:border-box`

前面提到，为盒子模型设置宽度，结果只是设置了内容的宽度，这个不合理。如何解决这一问题？答案就是为盒子指定样式：**`box-sizing:border-box`**。

```
<div style="padding:10px; border:5px solid blue; margin: 10px; width:300px;
box-sizing:border-box;">
    之前看过一篇文章，叫做《浏览器工作原理：新式网络浏览器幕后揭秘》
</div>
```

上图中，为`div`设置了`box-sizing:border-box`之后，300px 的**宽度是内容 + `padding` + 边框的宽度（不包括`margin`）**，这样就比较符合我们的实际要求了。建议大家在为系统写 CSS 时候，第一个样式是：

```
* {
    box-sizing:border-box;
}
```

大名鼎鼎的 Bootstrap 也把`box-sizing:border-box`加入到它的`*`选择器中，我们为什么不这样做呢？

纵向 margin 重叠

这里提到 margin，就不得不提一下 margin 的这一特性——纵向重叠。如`<p>`的纵向 margin 是 16px，那么两个`<p>`之间纵向的距离是多少？—— 按常理来说应该是 16 + 16 = 32px，但是答案仍然是 16px。因为纵向的 margin 是会重叠的，如果两者不一样大的话，大的会把小的“吃掉”。

浮动`float`

float 用于网页布局比较多，使用起来也比较简单，这里总结了一些比较重要、需要注意的知识点，供大家参考。

误解和误用

float 被设计出来的初衷是用于**文字环绕效果**，即一个图片一段文字，图片`float:left`之后，文字会环绕图片。

```
<div>
    <img src="image/1.png" style="float:left">
    一段文字一段文字一段文字一段文字一段文字一段文字一段文字一段文字一段文字
</div>
```

但是，后来大家发现结合`float + div`可以实现之前通过`table`实现的网页布局，因此就被“误用”于网页布局了。

> 题目：为何 float 会导致父元素塌陷?

破坏性

float 的**破坏性** —— float 破坏了父标签的原本结构，使得父标签出现了坍塌现象。导致这一现象的最根本原因在于：**被设置了 float 的元素会脱离文档流**。其根本原因在于 float 的设计初衷是解决文字环绕图片的问题。大家要记住 float 的这个影响。

包裹性

包裹性也是 float 的一个非常重要的特性，大家用 float 时一定要熟知这一特性。咱们还是先从一个小例子看起：

如上图，普通的 div 如果没有设置宽度，它会撑满整个屏幕，在之前的盒子模型那一节也讲到过。而如果给 div 增加`float:left`之后，它突然变得紧凑了，宽度发生了变化，把内容中的三个字包裹了——这就是包裹性。为 div 设置了 float 之后，其宽度会自动调整为包裹住内容宽度，而不是撑满整个父容器。

注意，此时 div 虽然体现了包裹性，但是它的 display 样式是没有变化的，还是`display: block`。

float 为什么要具有包裹性？其实答案还是得从 float 的设计初衷来寻找，float 是被设计用于实现文字环绕效果的。文字环绕图片比较好理解，但是如果想要让文字环绕一个 div 呢？此时 div 不被“包裹”起来的话，就无法实现环绕效果了。

清空格

float 还有一个大家可能不是很熟悉的特性——清空格。按照惯例，咱还是先举例子说明。

```
<div style="border: 2px solid blue; padding:3px;">
    <img src="image/1.png"/>
    <img src="image/2.png"/>
    <img src="image/3.png"/>
    <img src="image/4.png"/>
</div>
```

加上`float:left`之后：

上面第一张图中，正常的 img 中间是会有空格的，因为多个 img 标签会有换行，而浏览器识别换行为空格，这也是很正常的。第二张图中，为 img 增加了`float:left`的样式，这就使得 img 之间没有了空格，4 个 img 紧紧挨着。

如果大家之前没注意，现在想想之前写过的程序，是不是有这个特性。为什么 float 适合用于网页排版（俗称"砌砖头"）？就是因为 float 排版出来的网页严丝合缝，中间连个苍蝇都飞不进去。

"清空格"这一特性的根本原因是 float 会导致节点脱离文档流结构。它都不属于文档流结构了，那么它身边的什么换行、空格就都和它没了关系，它就尽量往一边靠拢，能靠多近就靠多近，这就是清空格的本质。

> 题目：手写 clearfix

`clearfix`

清除浮动的影响，一般使用的样式如下，统称`clearfix`代码。所有 float 元素的父容器，一般情况下都应该加`clearfix`这个 class。

```
.clearfix:after {
    content: '';
    display: table;
    clear: both;
}
.clearfix {
    *zoom: 1; /* 兼容 IE 低版本 */
}
```

```
<div class="clearfix">
    <img src="image/1.png" style="float: left"/>
    <img src="image/2.png" style="float: left"/>
</div>
```

小结

float 的设计初衷是解决文字环绕图片的问题，后来误打误撞用于做布局，因此有许多不合适或者需要注意的地方，上文基本都讲到了需要的知识点。如果是刚开始接触 float 的同学，学完上面的基础知识之后，还应该做一些练习实战一下 —— 经典的"圣杯布局"和"双飞翼布局"。这里就不再展开讲了，网上资料非常多，例如浅谈面试中常考的两种经典布局——圣杯与双飞翼（此文的最后两张图清晰地

展示了这两种布局）。

定位`position`

position 用于网页元素的定位，可设置 static/relative/absolute/fixed 这些值，其中 static 是默认值，不用介绍。

> 题目：relative 和 absolute 有何区别?

relative

相对定位 relative 可以用一个例子很轻松地演示出来。例如我们写 4 个`<p>`，出来的样子大家不用看也能知道。

```
<p>第一段文字</p>
<p>第二段文字</p>
<p>第三段文字</p>
<p>第四段文字</p>
```

然后我们在第三个`<p>`上面，加上`position:relative`并且设置`left`和`top`值，看这个`<p>`有什么变化。

```
<p>第一段文字</p>
<p>第二段文字</p>
<p style="position:relative; top: 10px; left: 10px">第三段文字</p>
<p>第四段文字</p>
```

上图中，大家应该要识别出两个信息（相信大部分人会忽略第二个信息）

- 第三个`<p>`发生了位置变化，分别向右向下移动了10px；
- 其他的三个`<p>`位置没有发生变化，这一点也很重要。

可见，**relative 会导致自身位置的相对变化，而不会影响其他元素的位置、大小**。这是 relative 的要点之一。还有第二个要点，就是 relative 产生一个新的定位上下文。下文有关于定位上下文的详细介绍，这里可以先通过一个例子来展示一下区别：

注意看这两图的区别，下文将有解释。

absolute

还是先写一个基本的 demo。

```
<p>第一段文字</p>
<p>第二段文字</p>
<p style="background: yellow">第三段文字</p>
<p>第四段文字</p>
```

然后，我们把第三个 `<p>` 改为 `position:absolute;`，看看会发生什么变化。

从上面的结果中，我们能看出几点信息：

- absolute 元素脱离了文档结构。和 relative 不同，其他三个元素的位置重新排列了。只要元素会脱离文档结构，它就会产生破坏性，导致父元素坍塌。（此时你应该能立刻想起来，float 元素也会脱离文档结构。）
- absolute 元素具有“包裹性”。之前 `<p>` 的宽度是撑满整个屏幕的，而此时 `<p>` 的宽度刚好是内容的宽度。
- absolute 元素具有“跟随性”。虽然 absolute 元素脱离了文档结构，但是它的位置并没有发生变化，还是老老实实地呆在它原本的位置，因为我们此时没有设置 top、left 的值。
- absolute 元素会悬浮在页面上方，会遮挡住下方的页面内容。

最后，通过给 absolute元素设置 top、left 值，可自定义其内容，这个都是平时比较常用的了。这里需要注意的是，设置了 top、left 值时，元素是相对于最近的定位上下文来定位的，而不是相对于浏览器定位。

fixed

其实 fixed 和 absolute 是一样的，唯一的区别在于：absolute 元素是根据最近的定位上下文确定位置，而 fixed 根据 window （或者 iframe）确定位置。

> 题目：relative、absolute 和 fixed 分别依据谁来定位?

定位上下文

relative 元素的定位永远是相对于元素自身位置的，和其他元素没关系，也不会影响其他元素。

fixed 元素的定位是相对于 window （或者 iframe）边界的，和其他元素没有关系。但是它具有破坏性，会导致其他元素位置的变化。

absolute 的定位相对于前两者要复杂许多。如果为 absolute 设置了 top、left，浏览器会根据什么去确定它的纵向和横向的偏移量呢？答案是浏览器会递归查找该元素的所有父元素，如果找到一个设置了 `position:relative/absolute/fixed` 的元素，就以该元素为基准定位，如果没找到，就以浏览器边界定位。如下两个图所示：

`flex` 布局

布局的传统解决方案基于盒子模型，依赖 `display` 属性 + `position` 属性 + `float` 属性。它对于那些特殊布局非常不方便，比如，垂直居中（下文会专门讲解）就不容易实现。在目前主流的移动端页面中，使用 flex 布局能更好地完成需求，因此 flex 布局的知识是必须要掌握的。

基本使用

任何一个容器都可以使用 flex 布局，代码也很简单。

```
<style type="text/css">
    .container {
      display: flex;
    }
    .item {
        border: 1px solid #000;
        flex: 1;
    }
</style>

<div class="container">
    <div class="item">aaa</div>
    <div class="item" style="flex: 2">bbb</div>
    <div class="item">ccc</div>
    <div class="item">ddd</div>
</div>
```

注意，第三个`<div>`的`flex: 2`，其他的`<div>`的`flex: 1`，这样第二个`<div>`的宽度就是其他的`<div>`的两倍。

设计原理

设置了`display: flex`的元素，我们称为“容器”（flex container），其所有的子节点我们称为“成员”（flex item）。容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做 main start，结束位置叫做 main end；交叉轴的开始位置叫做 cross start，结束位置叫做cross end。项目默认沿主轴排列。单个项目占据的主轴空间叫做 main size，占据的交叉轴空间叫做 cross size。

将以上文字和图片结合起来，再详细看一遍，这样就能理解 flex 的设计原理，才能更好地实际使用。

设置主轴的方向

`flex-direction`可决定主轴的方向，有四个可选值：

- row（默认值）：主轴为水平方向，起点在左端。
- row-reverse：主轴为水平方向，起点在右端。
- column：主轴为垂直方向，起点在上沿。
- column-reverse：主轴为垂直方向，起点在下沿。

```
.box {
  flex-direction: column-reverse| column | row | row-reverse;
}
```

以上代码设置的主轴方向，将依次对应下图：

设置主轴的对齐方式

`justify-content`属性定义了项目在主轴上的对齐方式，值如下：

- flex-start（默认值）：向主轴开始方向对齐。
- flex-end：向主轴结束方向对齐。
- center： 居中。
- space-between：两端对齐，项目之间的间隔都相等。
- space-around：每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

```
.box {
    justify-content: flex-start | flex-end | center | space-between |
space-around;
}
```

交叉轴的对齐方式

`align-items`属性定义项目在交叉轴上如何对齐，值如下：

- flex-start：交叉轴的起点对齐。
- flex-end：交叉轴的终点对齐。
- center：交叉轴的中点对齐。
- baseline: 项目的第一行文字的基线对齐。
- stretch（默认值）：如果项目未设置高度或设为 auto，将占满整个容器的高度。

```
.box {
    align-items: flex-start | flex-end | center | baseline | stretch;
}
```

如何实现居中对齐?

> 题目：如何实现水平居中?

水平居中

inline 元素用`text-align: center;`即可，如下：

```
.container {
   text-align: center;
}
```

block 元素可使用`margin: auto;`，PC 时代的很多网站都这么搞。

```
.container {
    text-align: center;
}
.item {
    width: 1000px;
    margin: auto;
}
```

绝对定位元素可结合`left`和`margin`实现，但是必须知道宽度。

```
.container {
    position: relative;
    width: 500px;
}
.item {
    width: 300px;
    height: 100px;
    position: absolute;
    left: 50%;
    margin: -150px;
}
```

> 题目：如何实现垂直居中?

垂直居中

inline 元素可设置`line-height`的值等于`height`值，如单行文字垂直居中：

```
.container {
   height: 50px;
   line-height: 50px;
}
```

绝对定位元素，可结合`left`和`margin`实现，但是必须知道尺寸。

- 优点：兼容性好
- 缺点：需要提前知道尺寸

```
.container {
    position: relative;
    height: 200px;
}
.item {
    width: 80px;
    height: 40px;
    position: absolute;
    left: 50%;
    top: 50%;
    margin-top: -20px;
    margin-left: -40px;
}
```

绝对定位可结合`transform`实现居中。

- 优点：不需要提前知道尺寸
- 缺点：兼容性不好

```
.container {
    position: relative;
    height: 200px;
}
.item {
    width: 80px;
    height: 40px;
    position: absolute;
    left: 50%;
    top: 50%;
    transform: translate(-50%, -50%);
    background: blue;
}
```

绝对定位结合`margin: auto`，不需要提前知道尺寸，兼容性好。

```
.container {
    position: relative;
    height: 300px;
}
.item {
    width: 100px;
    height: 50px;
    position: absolute;
    left: 0;
    top: 0;
    right: 0;
    bottom: 0;
    margin: auto;
```

```
}
```

其他的解决方案还有，不过没必要掌握太多，能说出上文的这几个解决方案即可。

理解语义化

> 题目：如何理解 HTML 语义化?

所谓“语义”就是为了更易读懂，这要分两部分：

- 让人（写程序、读程序）更易读懂
- 让机器（浏览器、搜索引擎）更易读懂

让人更易读懂

对于人来说，代码可读性、语义化就是一个非常广泛的概念了，例如定义 JS 变量的时候使用更易读懂的名称，定义 CSS class 的时候也一样，例如 `length` `list` 等，而不是使用 `a` `b` 这种谁都看不懂的名称。

不过我们平常考查的“语义化”并不会考查这么广义、这么泛的问题，而是考查 HTML 的语义化，是为了更好地让机器读懂 HTML。

让机器更易读懂

HTML 符合 XML 标准，但又和 XML 不一样 —— HTML 不允许像 XML 那样自定义标签名称，HTML 有自己规定的标签名称。问题就在这里 —— HTML 为何要自己规定那么多标签名称呢，例如 `p` `div` `h1` `ul` 等 —— 就是为了语义化。其实，如果你精通 CSS 的话，你完全可以全部用 `<div>` 标签来实现所有的网页效果，其他的 `p` `h1` `ul` 等标签可以一个都不用。但是我们不推荐这么做，这样做就失去了 HTML 语义化的意义。

拿搜索引擎来说，爬虫下载到我们网页的 HTML 代码，它如何更好地去理解网页的内容呢？—— 就是根据 HTML 既定的标签。`h1` 标签就代表是标题；`p` 里面的就是段落详细内容，权重肯定没有标题高；`ul` 里面就是列表；`strong` 就是加粗的强调的内容 ..... 如果我们不按照 HTML 语义化来写，全部都用 `<div>` 标签，那搜索引擎将很难理解我们网页的内容。

为了加强 HTML 语义化，HTML5 标准中又增加了 `header` `section` `article` 等标签。因此，书写 HTML 时，语义化是非常重要的，否则 W3C 也没必要辛辛苦苦制定出这些标准来。

CSS3 动画

CSS3 可以实现动画，代替原来的 Flash 和 JavaScript 方案。

首先，使用 `@keyframes` 定义一个动画，名称为 `testAnimation`，如下代码，通过百分比来设置不同的 CSS 样式，规定动画的变化。所有的动画变化都可以这么定义出来。

```
@keyframes testAnimation
{
    0%   {background: red; left:0; top:0;}
    25%  {background: yellow; left:200px; top:0;}
    50%  {background: blue; left:200px; top:200px;}
    75%  {background: green; left:0; top:200px;}
    100% {background: red; left:0; top:0;}
}
```

然后，针对一个 CSS 选择器来设置动画，例如针对 `div` 元素设置动画，如下：

```
div {
    width: 100px;
    height: 50px;
    position: absolute;

    animation-name: myfirst;
    animation-duration: 5s;
}
```

`animation-name` 对应到动画名称， `animation-duration` 是动画时长，还有其他属性：

- `animation-timing-function`：规定动画的速度曲线。默认是 `ease`
- `animation-delay`：规定动画何时开始。默认是 0
- `animation-iteration-count`：规定动画被播放的次数。默认是 1
- `animation-direction`：规定动画是否在下一周期逆向地播放。默认是 `normal`
- `animation-play-state`：规定动画是否正在运行或暂停。默认是 `running`
- `animation-fill-mode`：规定动画执行之前和之后如何给动画的目标应用，默认是 `none`，保留在最后一帧可以用 `forwards`

> 题目：CSS 的 `transition` 和 `animation` 有何区别?

首先 `transition` 和 `animation` 都可以做动效，从语义上来理解， `transition` 是过渡，由一个状态过渡到另一个状态，比如高度 `100px` 过渡到 `200px`；而 `animation` 是动画，即更专业做动效的， `animation` 有帧的概念，可以设置关键帧 `keyframe`，一个动画可以由多个关键帧多个状态过渡组成，另外 `animation` 也包含上面提到的多个属性。

重绘和回流

重绘和回流是面试题经常考的题目，也是性能优化当中应该注意的点，下面笔者简单介绍下。

- **重绘**：指的是当页面中的元素不脱离文档流，而简单地进行样式的变化，比如修改颜色、背景等，浏览器重新绘制样式
- **回流**：指的是处于文档流中 DOM 的尺寸大小、位置或者某些属性发生变化时，导致浏览器重新渲染部分或全部文档的情况

相比之下，**回流要比重绘消耗性能开支更大**。另外，一些属性的读取也会引起回流，比如读取某个 DOM 的高度和宽度，或者使用`getComputedStyle`方法。在写代码的时候要避免回流和重绘。比如在笔试中可能会遇见下面的题目：

> 题目：找出下面代码的优化点，并且优化它

```
var data = ['string1', 'string2', 'string3'];
for(var i = 0; i < data.length; i++){
    var dom = document.getElementById('list');
    dom.innerHTML += '<li>' + data[i] + '</li>';
}
```

上面的代码在循环中每次都获取`dom`，然后对其内部的 HTML 进行累加`li`，每次都会操作 DOM 结构，可以改成使用`documentFragment`或者先遍历组成 HTML 的字符串，最后操作一次`innerHTML`。

小结

本小节总结了 CSS 和 HTML 常考的知识点，包括 CSS 中比较重要的定位、布局的知识，也介绍了一些 CSS3 的知识点概念和题目，以及 HTML 的语义化。