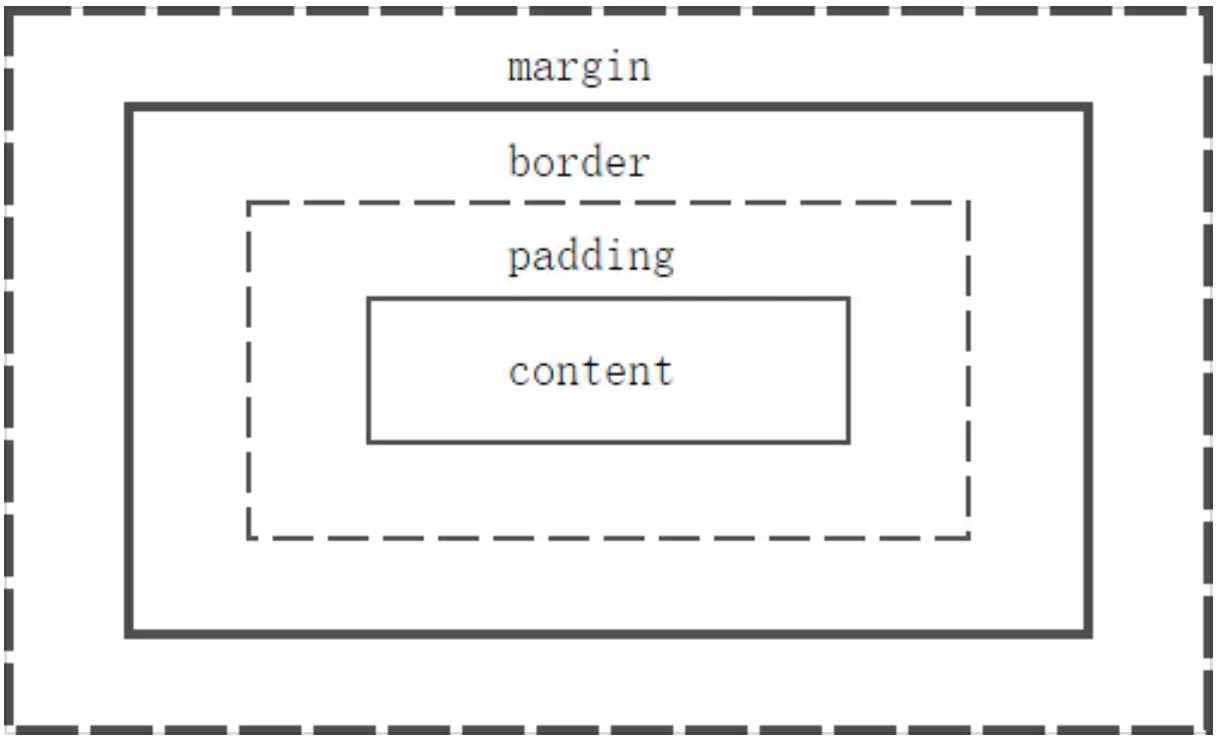
Basics

盒模型

CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：外边距（margin）、边框（border）、内边距（padding）、实际内容（content）四个属性。



CSS盒模型：**标准模型 + IE模型**
标准盒子模型：width/height=content
低版本IE盒子模型：width/height=content+padding+border

了解box-sizing吗？

- `content-box` 默认值，标准盒模型。任何边框和内边距的宽度都会被增加到最后绘制出来的元素宽度中。
- `border-box` I。E盒模型告诉浏览器：内容区的实际宽度是width减去(border + padding)的值。大多数情况下，这使得我们更容易地设定一个元素的宽高。

边界塌陷

当两个外边距相遇时，他们将形成一个外边距，合并后的外边距高度等于两个发生合并的外边距的高度中的较大者。
注意：只有普通文档流中块框的垂直外边距才会发生外边距合并，行内框、浮动框或绝对定位之间的外边距不会合并。

具体情况：

1. 浮动元素和其他任何元素之间不发生外边距叠加 (包括和它的子元素)。
2. 创建了 BFC 的元素不会和它的子元素发生外边距叠加。
3. 绝对定位元素和其他任何元素之间不发生外边距叠加(包括和它的子元素)。
4. inline-block 元素和其他任何元素之间不发生外边距叠加 (包括和它的子元素)。
5. 普通流中的块级元素的 margin-bottom 永远和它相邻的下一个块级元素的 margin-top 叠加（除非相邻的兄弟元素clear）。
6. 普通流中的块级元素（没有 border-top、没有 padding-top ）的 margin-top 和它的第一个普通流中的子元素（没有 clear ）发生 margin-top 叠加。
7. 普通流中的块级元素（ height 为 auto、min-height 为 0、没有 border-bottom、没有 padding-bottom ）和它的最后一个普通流中的子元素（没有自身发生 margin 叠加或 clear ）发生 margin-bottom 叠加。
8. 如果一个元素的 min-height 为 0、没有 border、没有 padding、高度为 0 或者 auto、不包含子元素，那么它自身的外边距会发生叠加。

总结一句：产生折叠的前提是**margin必须是邻接的**。

解决方法：
为父元素设置 BFC(包括overflow:auto) 或 padding 或 border (解决父子重叠)，兄弟元素间设置 float 或 inline-block 或 absolute(创建BFC不一定可以，设置overflow就不可以)。

常见定位方案

在讲 BFC 之前，我们先来了解一下常见的定位方案，定位方案是控制元素的布局，有三种常见方案:

普通流 (normal flow)

在普通流中，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。

浮动 (float)

在浮动布局中，元素首先按照普通流的位置出现，然后根据浮动的方向尽可能的向左边或右边偏移，其效果与印刷排版中的文本环绕相似。

绝对定位 (absolute positioning)

在绝对定位布局中，元素会整体脱离普通流，因此绝对定位元素不会对其兄弟元素造成影响，而元素具体的位置由绝对定位的坐标决定。

什么是 BFC（Block Formatting Context）

BFC是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。**具有 BFC 特性的元素可以看作是隔离了的独立容器，容器里面的元素不会在布局上影响到外面的元素，并且 BFC 具有普通容器所没有的一些特性。**

W3C定义：

> 浮动元素和绝对定位元素，非块级盒子的块级容器（例如 inline-blocks, table-cells, 和 table-captions），以及overflow值不为“visiable”的块级盒子，都会为他们的内容创建新的BFC（块级格式上下文）。
> BFC就是一种布局方式，在这种布局方式下，盒子们自所在的containing block顶部一个接一个垂直排列，水平方向上撑满整个宽度（除非内部盒子自己建立了新的BFC）。两个相邻的BFC之间的距离由margin决定。在同一个BFC内部，两个垂直方向相邻的块级元素的margin会发生“塌陷”。

形成 BFC 的五种条件：

- 浮动元素，float 除 none 以外的值
- 定位元素，position（absolute，fixed）
- display 为以下其中之一的值 inline-block，table-cell，table-caption
- overflow 除了 visible 以外的值（hidden，auto，scroll）
- HTML 就是一个 BFC

BFC 的特性：

- 内部的 Box 会在垂直方向上一个接一个的放置。
- 垂直方向上的距离由 margin 决定
- bfc 的区域不会与 float 的元素区域重叠。
- 计算 bfc 的高度时，浮动元素也参与计算
- bfc 就是页面上的一个独立容器，容器里面的子元素不会影响外面元素。

应用

1. 解决margin叠加问题，**如果想要避免外边距的重叠，可以将其放在不同的 BFC 容器中。**。
2. **BFC 可以包含浮动的元素（清除浮动），解决浮动元素父元素高度坍塌问题**
3. **BFC 可以阻止元素被浮动元素覆盖**，这个方法可以用来实现两列自适应布局

margin 外边距

margin中的top、right、bottom、left并不是一类，它们相对的参考线不一致。top 和 left 为一类，right和botom为一类。

margin数值为正的时候，移位的情况：

- top 以 containing block 的 content 上边或者垂直上方相连元素 margin 的下边为参考线垂直向下位移；
- left 以 containing block 的 content 左边或者水平左方相连元素 margin 的右边为参考线水平向右位移。
- right 以元素本身的 border 右边为参考线水平向右位移；
- bottom 以元素本身的border 下边为参考线垂直向下位移。

margin数值为负的时候，与上面相反，移位的情况：
可以理解为 left 和 top 是紧靠着已经布局好了的元素，它们的位置是不会改变的，所以 left 和 top 是相对于其他元素，而 right 和 bottom 是相对于自身。
box 最后的显示大小等于 box 的 border 及 border 内的大小加上正的 margin 值。而负的 margin 值不会影响 box 的实际大小，如果是负的 top 或 left 值会引起 box 的向上或向左位置移动，如果是 bottom 或 right 只会影响下面 box 的显示的参考线。
margin数值为负的另外一个作用：
当元素不存在width属性或者（width：auto）的时候，负margin会增加元素的宽度，margin-left和margin-right都是可以增加宽度。margin-top为负值不会增加高度，只会产生向上位移。margin-bottom为负值不会产生位移，会减少自身的供css读取的高度。

负边距在让元素产生偏移时和position: relative**有什么区别?**
负margin和position: relative在让元素产生偏移时都没有脱离文档流。但是利用负margin让元素产生偏移时，元素原来的位置会被占据；而position: relative元素原来的位置不会被占据，仍然会被保留。

笔试

圣杯布局

1. **通过 float + margin**

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

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
        * {
            margin: 0;
```

```
        padding: 0;
    }

    html,
    body {
        width: 100%;
        height: 100%;
    }

    .wrapper {
        padding: 0 150px 0 200px;
    }

    .main,
    .left,
    .right {
        float: left;
    }

    .main {
        width: 100%;
        background: blue;
        height: 300px;
    }

    .left {
        position: relative;
        left: -200px;
        width: 200px;
        height: 300px;
        margin-left: -100%;
        background: red;
    }

    .right {
        position: relative;
        right: -150px;
        width: 150px;
        height: 300px;
```

```
            /* 右边栏自身的宽度 */
            margin-left: -150px;
            background: orange;
        }
    </style>
</head>

<body>
    <div class="wrapper">
        <div class="main"></div>
        <div class="left"></div>
        <div class="right"></div>
    </div>
</body>

</html>
```

2. 通过flex

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

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
        html,
        body {
            height: 100%;
        }

        div {
            height: 100%;
        }

        .container {
            display: flex;
        }
```

```
        .content {
            flex: 1 1;
            order: 2;
            background: #f00;
        }

        .left {
            flex: 0 0 200px;
            order: 1;
            background: #0f0;
        }

        .right {
            flex: 0 0 300px;
            order: 3;
            background: #00f;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="content">圣杯布局</div>
        <div class="left">左</div>
        <div class="right">右</div>
    </div>
</body>

</html>
```

双飞翼布局

```
<!DOCTYPE html>
<html>

<head>
```

```
    <meta charset="utf-8">
    <title>双飞翼布局</title>
    <style>
        html,
        body,
        div {
            margin: 0px;
            padding: 0px;
        }

        .header,
        .footer {
            border: 1px solid;
            background: #ccc;
            text-align: center;
        }

        .wrapper:after {
            content: "";
            display: block;
            clear: both;
        }

        .middle {
            width: 100%;
            height: 300px;
            background: blue;
            float: left;
        }

        .middle .main {
            margin: 0px 150px 0px 200px;
        }

        .sub {
            width: 200px;
            height: 300px;
            background: red;
            float: left;
```

```
            margin-left: -100%;
        }

        .extra {
            width: 150px;
            height: 300px;
            background: orange;
            float: left;
            margin-left: -150px;
        }
    </style>
</head>

<body>
    <div class="header">header</div>
    <div class="wrapper">
        <div class="middle">
            <div class="main">main</div>
        </div>
        <div class="sub">sub</div>
        <div class="extra">extra</div>
    </div>
    <div class="footer">footer</div>
</body>

</html>
```

CSS 的权重和优先级

权重

行内样式+1000

id选择器+100

属性选择器、class或者伪类+10,

元素选择器

伪元素+1

通配符+0

优先级

- 权重相同，写在后面的覆盖前面的
- 使用 !important 达到最大优先级，都使用 !important 时，权重大的优先级高

Flex布局

介绍 Flex 布局，flex 是什么属性的缩写：

- 弹性盒布局，CSS3 的新属性，用于方便布局，比如垂直居中
- flex属性是 flex-grow、flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto

垂直水平居中

水平居中

- 行内元素水平居中
  把行内元素包裹在父元素（ 、<li>、<p>等）中，并且在父元素设置，适用于文字，链接，及其inline或者inline-block、inline-table和inline-flex。

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

- 块状元素的水平居中
  设置左右margin为auto，元素设置：

```
#center {
    margin: 0 auto;
}
```

flex布局

```
#center {
    display: flex;
    // display: inline-flex
    justify-content: center
}
```

垂直水平居中

- 已知高度宽度元素的水平垂直居中
  1. 方法1: 利用绝对定位+负margin。

```
#container {
    display: relative;
}

#item {
    width: 100px;
    height: 100px;
    position: absolute;
    top: 50%;
    left: 50%;
    margin: -50px -50px;
}
```

  2. 方法2: 绝对定位+margin

```
#container {
    position: relative;
}
#center {
    position: absolute;
    margin: auto;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
}
```

- 未知宽高元素水平垂直居中
  1. flex布局

```
#container {
```

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

2. 方法5: 用js控制

```
var parent = document.getElementById("parent");
var current = document.getElementById("current");
current.style.top = (parent.offsetHeight - current.offsetHeight) / 2 + "px";
current.style.left = (parent.offsetWidth - current.offsetWidth) / 2 + "px";
```

3. CSS3的 transform

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

4. 利用表格元素，将父元素设置为display: table，子元素设置为table-cell

```
#wrapper {
    display: table;
    text-align: center;
}
#cell {
    display: table-cell;
    text-align: middle;
}
```

说一下你知道的position属性，都有啥特点？

- static：无特殊定位，对象遵循正常文档流。top，right，bottom，left等属性不会被应用。
- relative：对象遵循正常文档流，但将依据top，right，bottom，left等属性在正常文档流中偏移位置。而其层叠通过z-index属性定义。
- absolute：对象脱离正常文档流，使用top，right，bottom，left等属性进行绝对定位。而其层叠通过z-index属性定义。
- fixed：对象脱离正常文档流，使用top，right，bottom，left等属性以窗口为参考点进行定位，当出现滚动条时，对象不会随着滚动。而其层叠通过z-index属性定义。
- sticky：具体是类似 relative 和 fixed，在 viewport 视口滚动到阈值之前应用 relative，滚动到阈值之后应用 fixed 布局，由 top 决定。

清除浮动有哪些方法？

不清除浮动会发生高度塌陷：浮动元素父元素高度自适应（父元素不写高度时，子元素写了浮动后，父元素会发生高度塌陷）

- clear清除浮动（添加空div法）在浮动元素下方添加空div,并给该元素写css样式

```
{
  clear: both;
  height: 0;
  overflow: hidden;
}
```

- 给浮动元素父级设置高度
- 父级同时浮动（需要给父级同级元素添加浮动）
- 父级设置成 `inline-block`，其 `margin: 0 auto` 居中方式失效
- 给父级添加 `overflow:hidden` 清除浮动方法
- 万能清除法 after伪类 清浮动（现在主流方法，推荐使用）

```
.float_div:after {
  content: ".";
  clear: both;
  display: block;
  height: 0;
```

```
  overflow: hidden;
  visibility: hidden;
}
.float_div{
  zoom: 1
}
```

CSS动画

transition

作用：用于设置元素的样式过度，和animation有着类似的效果，但细节上有很大的不同。

语法：**transition: property duration timing-function delay;**

值	描述
transition-property	规定设置过渡效果的 CSS 属性的名称
transition-duration	规定完成过渡效果需要多少秒或毫秒
transition-timing-function	规定速度效果的速度曲线
transition-delay	定义过渡效果何时开始

实例：

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

```
    <title>transition</title>
    <style>
        #box {
            height: 100px;
            width: 100px;
            background: red;
            transition: transform 1s ease-in 1s;
        }

        #box:hover {
            transform: rotate(180deg) scale(.5, .5);
        }
    </style>
</head>
<body>
    <div id="box"></div>
</body>
</html>
```

transition产生动画的条件是transition设置的property发生变化，这种动画的特点是需要"一个驱动力去触发"，有着以下几个不足：

1. 需要事件触发，所以没法在网页加载时自动发生
2. 是一次性的，不能重复发生，除非一再触发
3. 只能定义开始状态和结束状态，不能定义中间状态，也就是说只有两个状态
4. 一条transition规则，只能定义一个属性的变化，不能涉及多个属性。

transform

用于元素进行旋转、缩放、移动或倾斜，和设置样式的动画并没有什么关系

translate

translate只是transform的一个属性值，即移动，除此之外还有 scale 等

animation

作用：用于设置动画属性，他是一个简写的属性，包含6个属性。animation与transition 不同的是，keyframes提供更多的控制，尤其是时间轴的控制，这点让css animation更加强大，使得flash的部分动画效果可以由css直接控制完成，而这一切，仅仅只需要几行代码，也因此诞生了大量基于css的动画库，用来取代flash的动画部分。

语法：**animation: name duration timing-function delay iteration-count direction play-state fill-mode;**

值	描述
name	用来调用@keyframes定义好的动画，与@keyframes定义的动画名称一致
duration	指定元素播放动画所持续的时间
timing-function	规定速度效果的速度曲线，是针对每一个小动画所在时间范围的变换速率
delay	定义在浏览器开始执行动画之前等待的时间，值整个animation执行之前等待的时间
iteration-count	定义动画的播放次数，可选具体次数或者无限(infinite)
direction	设置动画播放方向：normal(按时间轴顺序),reverse(时间轴反方向运行),alternate(轮流，即来回往复进行),alternate-reverse(动画先反运行再正方向运行，并持续交替运行)
play-state	控制元素动画的播放状态，通过此来控制动画的暂停和继续，两个值：running(继续)，paused(暂停)
fill-mode	控制动画结束后，元素的样式，有四个值：none(回到动画没开始时的状态)，forwards(动画结束后动画停留在结束状态)，backwords(动画回到第一帧的状态)，both(根据animation-direction轮流应用forwards和backwards规则)，注意与iteration-count不要冲突(动画执行无限次)

实例：

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

<head>
  <title>animation</title>
  <style>
    .box {
      height: 100px;
      width: 100px;
      border: 15px solid black;
      animation: changebox 1s ease-in-out 1s infinite alternate running
forwards;
    }

    .box:hover {
      animation-play-state: paused;
    }
```

```
    @keyframes changebox {
      10% {
        background: red;
      }
      50% {
        width: 80px;
      }
      70% {
        border: 15px solid yellow;
      }
      100% {
        width: 180px;
        height: 180px;
      }
    }
  </style>
</head>

<body>
  <div class="box"></div>
</body>

</html>
```

浏览器的回流与重绘 (Reflow & Repaint)

在讨论回流与重绘之前，我们要知道：

1. 浏览器使用流式布局模型 (Flow Based Layout)。
2. 浏览器会把HTML解析成DOM，把CSS解析成CSSOM，DOM和CSSOM合并就产生了Render Tree。
3. 有了RenderTree，我们就知道了所有节点的样式，然后计算他们在页面上的大小和位置，最后把节点绘制到页面上。
4. 由于浏览器使用流式布局，对Render Tree的计算通常只需要遍历一次就可以完成，但table及其内部元素除外，他们可能需要多次计算，通常要花3倍于同等元素的时间，这也是为什么要避免使用table布局的原因之一。

一句话：回流必将引起重绘，重绘不一定会引起回流。

回流（Reflow）

当Render Tree中部分或全部元素的尺寸、结构、或某些属性发生改变时，浏览器重新渲染部分或全部文档的过程称为回流。

会导致回流的操作：

- 页面首次渲染
- 浏览器窗口大小发生改变
- 元素尺寸或位置发生改变
- 元素内容变化（文字数量或图片大小等等）
- 元素字体大小变化
- 添加或者删除**可见**的DOM元素
- 激活CSS伪类（例如：:hover）
- 查询某些属性或调用某些方法

一些常用且会导致回流的属性和方法：

- clientWidth、clientHeight、clientTop、clientLeft
- offsetWidth、offsetHeight、offsetTop、offsetLeft
- scrollWidth、scrollHeight、scrollTop、scrollLeft
- scrollIntoView()、scrollIntoViewIfNeeded()
- getComputedStyle()
- getBoundingClientRect()
- scrollTo()

重绘（Repaint）

当页面中元素样式的改变并不影响它在文档流中的位置时（例如：color、background-color、visibility等），浏览器会将新样式赋予给元素并重新绘制它，这个过程称为重绘。

性能影响

回流比重绘的代价要更高。

现代浏览器会对频繁的回流或重绘操作进行优化：

浏览器会维护一个队列，把所有引起回流和重绘的操作放入队列中，如果队列中的任务数量或者时间间隔达到一个阈值的，浏览器就会将队列清空，进行一次批处理，这样可以把多次回流和重绘变成一次。

如何避免

CSS

- 避免使用table布局。
- 尽可能在DOM树的最末端改变class。

- 避免设置多层内联样式。
- 将动画效果应用到position属性为absolute或fixed的元素上。
- 避免使用CSS表达式（例如：calc()）。

JavaScript

- 避免频繁操作样式，最好一次性重写style属性，或者将样式列表定义为class并一次性更改class属性。
- 避免频繁操作DOM，创建一个documentFragment，在它上面应用所有DOM操作，最后再把它添加到文档中。
- 也可以先为元素设置display: none，操作结束后再把它显示出来。因为在display属性为none的元素上进行的DOM操作不会引发回流和重绘。
- 避免频繁读取会引发回流/重绘的属性，如果确实需要多次使用，就用一个变量缓存起来。
- 对具有复杂动画的元素使用绝对定位，使它脱离文档流，否则会引起父元素及后续元素频繁回流。

笔试

div 高度永远是宽度的一半?

```
<!-- css div 垂直水平居中，并完成 div 高度永远是宽度的一半（宽度可以不指定）？ -->

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

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
        }

        html,
        body {
            width: 100%;
            height: 100%;
        }
```

```
        .outer {
            height: 100%;
            background: blue;
            display: flex;
            align-items: center;
            justify-content: center;
        }

        .inner {
            position: relative;
            width: 100%;
            height: 0;
            padding-bottom: 50%;
            background: red;
        }

        .box {
            position: absolute;
            width: 100%;
            height: 100%;
        }
    </style>
</head>

<body>
    <div class="outer">
        <div class="inner">
            <div class="box">hello</div>
        </div>
    </div>
</body>

</html>
```

CSS 怎么画一个大小为父元素宽度一半的正方形?

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

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
        .outer {
            width: 400px;
            height: 600px;
            background: red;
        }

        .inner {
            width: 50%;
            padding-bottom: 50%;
            background: blue;
        }
    </style>
</head>

<body>
    <div class="outer">
        <div class="inner"></div>
    </div>
</body>

</html>
```

实现一个宽高自适应正方形?

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
```

```
    <style>
      .outer {
        padding-top: 50%;
        height: 0;
        background: #ccc;
        width: 50%;
        position: relative;
      }

      .inner {
        position: absolute;
        width: 100%;
        height: 100%;
        top: 0;
        left: 0;
        background: blue;
      }
    </style>
  </head>
  <body>
    <div class="outer">
      <div class="inner">hello</div>
    </div>
  </body>
</html>
```

参考文献

- 字节跳动最爱考的前端面试题：CSS 基础 - 掘金
- 头条笔试
- 负边距、三栏布局 - 简书
- Flex 布局教程：语法篇 - 阮一峰的网络日志
- CSS动画：animation、transition、transform、translate傻傻分不清 - 掘金
- 10 分钟理解 BFC 原理 - 知乎
- 浏览器的回流与重绘 (Reflow & Repaint) - 掘金