前言

盒模型是CSS中最重要最核心的概念，整个概念基本上环绕在CSS整个体系里，所有的样式和排版都基于该概念进行着。理解这个重要的概念才能更好地 排版页面 和 布局页面 ，后面遇到的排版和布局困难也能快速定位出问题的所在。

不要觉得简单而去忽略它们，往往越简单的东西蕴含的威力越大。换成人生道理就是说，不要小看一个人，TA虽然穿着短裤拖鞋，背后可能是拥有10栋出租楼的包租公或包租婆。

盒模型

盒模型又名**框模型**，是一种网页设计思维模型，它把文档节点看成一个盒子。

在HTML文档解析过程中，每个节点都会被描述为一个盒模型，然后一个盒子套进另一个盒子中，再依据各个节点对应的CSS规则，最后渲染成一个井井有条的页面。一个完整的页面就像以下的 惊喜爆炸礼盒 ，层层嵌套，直到最后一层出现惊喜为止。有女友的，赶紧买一个试试！

组成

盒模型由以下属性组成，由外到内用公式表示就是： `box = margin + border + padding + content` 。除了 `content` (不是属性，作为盒模型扩展理解使用)，其余属性都包含 `left` 、 `right` 、 `top` 和 `bottom` 等扩展属性。

- **margin**：边距，外部透明区域，负责隔离相邻盒子
- **border**：边框，内部着色区域，负责隔离边距和填充，包含 `width` 、 `style` 、 `color` 三个扩展属性
- **padding**：填充，内部着色区域，负责扩展盒子内部尺寸
- **content**：内容，以 文本 或 节点 存在的占用位置

重点提醒， `padding` 着色随 `background-color` 而变，可用 `background-clip` 隔离，该情况在第10章**背景与遮罩**中会详细讲解。

> 理解

节点由外到内一层一层深入，通过上述公式组成了一个完整的**盒模型**。所以在理解盒模型时记住这4个属性及其从外到内的顺序即可。换另一种方式理解，可把它看做你的快递包裹。两个快递包裹间的距离就是 `margin` ，快递包裹的纸皮就是 `border` ，打开快递包裹，填充物料就是 `padding` ，把填充物料打开看到了你的物品，那就是 `content` 。这样理解是不是特别容易呢?

类型

由于历史原因，盒模型分化成两种类型，分别是**标准盒模型**和**怪异盒模型**。具体原因在第2章**浏览器**有提及。

所以CSS3里提供一个属性用于声明盒模型的类型，它就是 `box-sizing` 。

- **content-box**：标准盒模型(默认)
- **border-box**：怪异盒模型

它不具备继承性，若全局统一盒模型，那只能使用 `*` 声明 `box-sizing` 了。建议使用reset.css里的方式声明。

> 标准盒模型

标准盒模型是W3C规范的标准，由 `margin + border + padding + content` 组成。与上述提到的公式一模一样，节点的 `width/height` 只包含 `content` ，不包含 `padding` 和 `border` 。

节点的尺寸计算公式如下。

- **横向**： `margin-[left/right] + border-[left/right] + padding-[left/right] + width`
- **纵向**： `margin-[top/bottom] + border-[top/bottom] + padding-[top/bottom] + height`

节点的宽高计算公式如下。

- **横向**： `width = width`
- **纵向**： `height = height`

> 怪异盒模型

怪异盒模型又名 `IE盒子模型` ，是 `IExplore` 制定的标准，由 `margin + content` 组成。与上述提到的公式一不同，节点的 `width/height` 包含 `border` 、 `padding` 和 `content` 。

节点的尺寸计算公式如下。

- **横向**： `margin-[left/right] + width` (包含 `border-[left/right]` 和 `padding-[left/right]`)
- **纵向**： `margin-[top/bottom] + height` (包含 `border-[top/bottom]` 和 `padding-[top/bottom]`)

节点的宽高计算公式如下。

- **横向**： `width = border + padding + width`
- **纵向**： `height = border + padding + height`

在IExplore中，若HTML文档缺失<!doctype html>声明则会触发怪异盒模型

> 两者区别

通过代码演示可能会更清晰， `width` 和 `height` 的范围也一目了然，其实两者区别在于 `width` 和 `height` 包不包含 `border` 和 `padding` 。把上述公式记清楚，两者区别就迎刃而解了。

```
.content-box {
    box-sizing: content-box;
    margin: 100px;
    padding: 50px;
    border: 10px solid #66f;
    width: 80px;
    height: 80px;
    background-color: #f66;
}
.border-box {
    box-sizing: border-box;
    margin: 100px;
    padding: 50px;
    border: 10px solid #66f;
    width: 80px;
    height: 80px;
    background-color: #f66;
}
```

视觉格式化模型

上述**盒模型**都是平时了解到的概念，若使用 `display` 对这个简单盒模型稍微加工则会进化到**视觉格式化模型**。

视觉格式化模型指在视觉媒体上处理和显示文档而使用的计算规则。它是一种CSS机制，由大量CSS规范组成，规定了节点在页面中的排版。

块级元素

当节点的 `display` 声明为 `block` 、 `list-item` 、 `table` 、 `flex` 或 `grid` 时，该节点被标记为**块级元素**。块级元素默认宽度为 `100%` ，在垂直方向上按顺序放置，同时参与**块格式化上下文**。

每个 块级元素 都至少生成一个块级盒，或一个块容器盒， 块级盒 描述它与兄弟节点间的表现方式， 块容器盒 描述它与子节点间的表现方式。

一个块容器盒只包含其他块级盒，或生成一个 行内格式化上下文 只包含行内盒。或许一段代码中某一个块容器盒同时包含块级盒和行内盒的情况，但实质上在这种情况下会产生一种新的匿名块盒解决该问题。

行内元素

当节点的 `display` 声明为 `inline` 、 `inline-block` 、 `inline-table` 、 `inline-flex` 或 `inline-grid` 时，该节点被标记为**行内元素**。行内元素默认宽度为 `auto` ，在水平方向上按顺序放置，同时参与**行内格式化上下文**。

当行内级盒参与 行内格式化上下文 后就会变成行内盒。另外还有一个叫做匿名行内盒的概念，匿名行内盒与匿名块盒的原理类似，都是浏览器自动生成的补充性盒，简单来一段代码理解理解匿名行内盒是如何产生的。

```
<p>我是<span>JowayYoung</span>，我的公众号是<span>IQ前端</span></p>
```

此时 我是 和 ，我的公众号是 就会生成了一个匿名行内盒，然后与两个 `<span></span>` 一起处于 `<p>` 参与 行内格式化上下文 后的行内盒中，并保持水平排列。

> 两者区别

上述概念可能有点绕口，若从两者的区别理解可能更容易消化。

- 互相转换
  - 块级元素转换行内元素： `display:inline`
  - 行内元素转换块级元素： `display:block`
- 占位表现
  - 块级元素默认独占一行，默认宽度为父元素的100%，可声明边距、填充、宽高
  - 行内元素默认不独占一行(一行可多个)，默认宽度随内容自动撑开，可声明水平边距和填充，不可声明垂直边距和宽高
- 包含关系
  - 块级元素可包含块级元素和行内元素
  - 行内元素可包含行内元素，不能包含块级元素

格式化上下文

概念内容多次提到了 格式化上下文 的字眼，那么 格式化上下文 又是何方神圣呢？了解 格式化上下文 ，或许就能了解上述内容了。

格式化上下文指决定渲染区域里节点的排版、关系和相互作用的渲染规则。简单来说就是页面中有一个 `<ul>` 及其多个子节点 `<li>` ， 格式上下文 决定这些 `<li>` 如何排版， `<li>` 与 `<li>` 间处于什么关系，以及 `<li>` 与 `<li>` 间如何互相影响。

格式上下文 由以下几部分组成，其中最重要的是**块格式化上下文**和**行内格式化上下文**，也频繁出现在大厂面试题中，了解其原理与特性，相信面试时被问到也无什么难度了。

上下文	缩写	版本	声明
块格式化上下文	BFC	2	块级盒子容器
行内格式化上下文	IFC	2	行内盒子容器
弹性格式化上下文	FFC	3	弹性盒子容器
格栅格式化上下文	GFC	3	格栅盒子容器

读中文读起来有点拗口，干脆读英文缩写好了，有兴趣的同学自行查阅MDN了解其英文单词

为了防止有些同学对 格式化上下文 的概念越看越混乱，本章不会过多解说 格式化上下文 ，但是笔者会抽丝剥茧把 格式化上下文 的概念清晰化，更多的解读可自行查找相关资料深入学习。有时学习点到即止也未尝不是一件好事，过多解读反而达不到想要的效果。

块格式化上下文

BFC是页面上一个独立且隔离的渲染区域，容器里的子节点不会在布局上影响到外面的节点，反之亦然。

以下是笔者意译W3C文档和平时一些开发经验的总结所得，也结合一些自身对BFC的理解。

> 规则

- 子节点在垂直方向上按顺序放置
- 子节点的垂直方向距离由 `margin` 决定，相邻节点的 `margin` 会发生重叠，以最大 `margin` 为合并值
- 每个节点的 `margin-left/right` 与父节点的 左边/右边 相接触，即使处于浮动也如此，除非自行形成BFC
- BFC区域不会与同级浮动区域重叠
- BFC是一个隔离且不受外界影响的独立容器
- 计算BFC高度时其浮动子节点也参与计算

> 成因

- 根节点： `html`
- 非溢出可见节点： `overflow:!visible`
- 浮动节点： `float:left/right`

- 绝对定位节点： `position:absolute/fixed`
- 被定义成块级的非块级节点： `display:inline-block/table-cell/table-caption/flex/inline-flex`
- 父节点与正常文档流的子节点(非浮动)自动形成BFC

> 场景

- 清除浮动
- 已知宽度水平居中
- 防止浮动节点被覆盖
- 防止垂直margin合并

面试中常问到的 `margin塌陷` 问题，可用BFC的概念回答了。所谓的塌陷其实是两个BFC的相邻盒或父子盒相互作用时产生的效果，两个盒子会取相邻边最大 `margin` 作为相邻边的共用 `margin` 。

在此笔者补充一些 `margin折叠` 的计算问题，相信在笔试上会遇到。

- 两个盒子相邻边的 `margin` 都为正值，取最大值
- 两个盒子相邻边的 `margin` 都为负值，取最小值，两者会互相重合
- 两个盒子相邻边的 `margin` 一正一负，取两者相加值，若结果为负，两者会互相重合

行内格式化上下文

IFC的宽高由行内子元素中最大的实际高度确定，不受垂直方向的 `margin` 和 `padding` 影响。另外，IFC中不能存在块元素，若插入块元素则会产生对应个数的匿名块并互相隔离，即产生对应个数的IFC，每个IFC对外表现为块级元素，并垂直排列。

以下是笔者意译W3C文档和平时一些开发经验的总结所得，也结合一些自身对IFC的理解。

> 规则

- 节点在水平方向上按顺序放置
- 节点无法指定宽高，其 `margin` 和 `padding` 在水平方向有效在垂直方向无效
- 节点在垂直方向上以不同形式对齐
- 节点的宽度由包含块和浮动决定，高度由行高决定

> 成因

- 行内元素： `display:inline[-x]`
- 声明 `line-height`
- 声明 `vertical-align`
- 声明 `font-size`

弹性格式化上下文

声明 `display` 为 `flex` 或 `inline-flex` 时，节点会生成一个 `FFC` 的独立容器，主要用于 响应式布局 。

格栅格式化上下文

声明 `display` 为 `grid` 或 `inline-grid` 时，节点会生成一个 `GFC` 的独立容器，主要用于 响应式布局 。

细心的同学会发现， `GFC` 有点像 `<table>` ，同为二维表格，但是 `GFC` 会有更丰富的属性控制行列、对齐以及更为精细的渲染语义和控制。不过由于兼容性不是特别好，所以笔者在本小册也不会讲解基于 `GFC` 的**格栅布局**。

文档流

文档流指节点在排版布局过程中默认使用从左往右从上往下的流式排列方式。在窗体自上而下分成一行行，且每行按照从左至右的顺序排列节点，其显著特点就是 从左往右从上往下 。

类型

对于一个标准的文档流，可根据其特性对节点分类。

- HTML级别
  - 容器级元素： `<div>` 、 `<ul>` 、 `<li>` 等
  - 文本级元素： `<a>` 、 `<p>` 、 `<span>` 等
- CSS级别
  - 块级元素： `<div>` 、 `<ul>` 、 `<li>` 等
  - 行内元素： `<a>` 、 `<p>` 、 `<span>` 等

微观现象

即使是标准的文档流，也不排除有一些小小的缺陷，笔者罗列三个常见缺陷。

- **空白折叠**：HTML中换行编写行内元素，排版会出现 5px空隙
- **高矮不齐**：行内元素统一以底边垂直对齐
- **自动换行**：排版若一行无法完成则换行接着排版

> 空白折叠解决方式

空白折叠也许是最容易出现的文档流微观现象，可能各位同学都会遇过。

```
<ul>
    <li></li>
    <li></li>
```

```
    <li></li>
</ul>


ul {
    text-align: center;
}
li {
    display: inline-block;
}
```

此时，很多浏览器就会出现 5px空隙 ，解决方式也有很多种的。

第一种，必须紧密连接节点。

```
<ul>
    <li></li><li></li><li></li>
</ul>
```

第二种，子节点声明 margin-left:-5px 。

```
li {
    display: inline-block;
    margin-left: -5px;
}
```

第三种，使用 Flex布局 居中显示。

```
ul {
    display: flex;
    justify-content: center;
}
```

脱流文档流

脱流文档流指节点脱流正常文档流后，在正常文档流中的其他节点将忽略该节点并填补其原先空间。文档一旦脱流，计算其父节点高度时不会将其高度纳入，脱流节点不占据空间，因此添加浮动或定位后会对周

围节点布局产生或多或少的影响。

文档流的脱流有两种方式。

- [x] 浮动布局： `float:left/right`
- [x] 定位布局： `position:absolute/fixed`

> Float方式

节点使用 `float` 脱流时，会让其跳出正常文档流，其他节点会忽略该节点并填补其原先空间。但是该节点文本可不参与这个脱流效果，却会认同该节点所占据的空间并围绕它布局，这个就是常说的 文字环绕效果 的原理。

一句话概括： 节点参与浮动布局后，自身脱流但其文本不脱流 。

> Position方式

节点使用 `position` 脱流时(只有 `absolute` 和 `fixed`)，会让其及其文本一起跳出正常文档流，其他节点会忽略该节点并填补其原先空间。 `absolute` 绝对定位是相对往上遍历第一个包含 `position:relative` 的祖先节点定位，若无此节点则相对 `<body>` 定位； `fixed` 固定定位是相对浏览器窗口定位。

一句话概括： 节点参与定位布局后，自身及其文本一起脱流 。

显隐影响

在正常文档流排版过程中，经常会使用 `display:none` 和 `visibility:hidden` 控制节点的隐藏， `display:none` 简称 `DN` ， `visibility:hidden` 简称 `VH` 。上一章有提及 `DN` 和 `VH` 的区别，这次看看节点切入隐藏状态后，会存在什么差别。

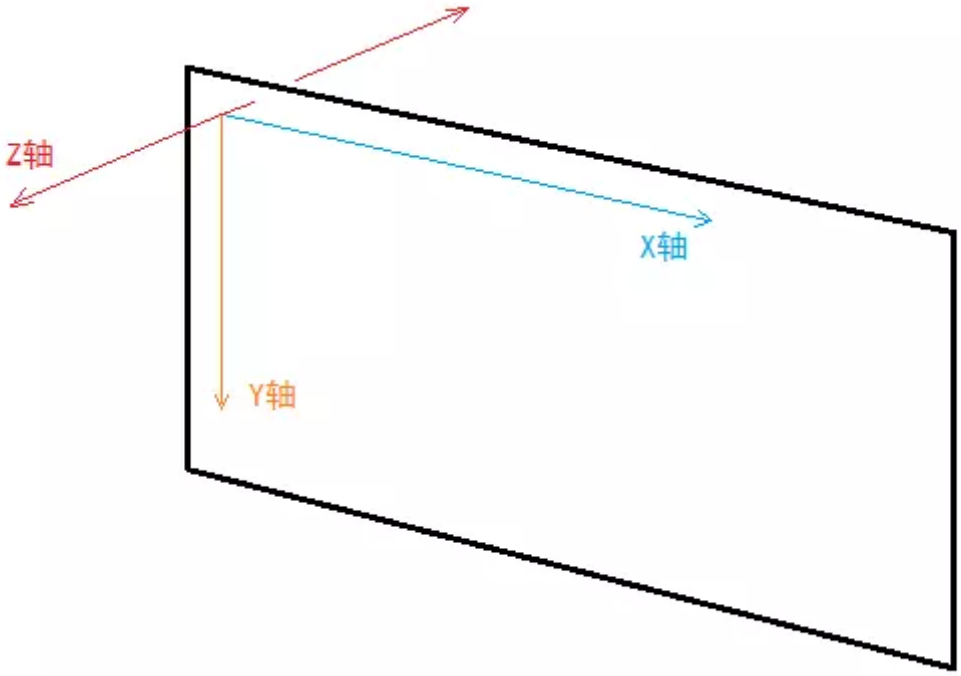
- 节点不可见但占据空间，显隐时可过渡： `visibility:hidden`
- 节点不可见但占据空间，不可点击： `visibility:hidden`
- 节点不可见不占据空间，可访问DOM： `display:none`
- 节点不可见但占据空间，可点击： `opacity:0`
- 节点不可见不占据空间，可点击： `position:absolute; opacity:0`
- 节点不可见但占据空间，不可点击： `position:relative; z-index:-1`
- 节点不可见不占据空间，不可点击： `position:absolute; z-index:-1`

当然这个问题也经常在大厂面试题中出现，结合 `DN` 和 `VH` 的区别，相信就能完美解答面试官的问题了。

层叠上下文

层叠上下文指盒模型在三维空间 `Z轴` 上所表现的行为。每个盒模型存在于一个三维空间中，分别是平面画布的 `X轴Y轴` 和表示层叠的 `Z轴` 。

通常情况下，节点在页面上沿着X轴和Y轴平铺，很难察觉它们在Z轴上的层叠关系。一旦节点发生堆叠，最终表现就是 节点间互相覆盖 。若一个节点包含 层叠上下文 ，那么该节点就拥有绝对的制高点，用一个成语贴切表示就是 鹤立鸡群 ，最终表现就是 离屏幕观察者更近 。



此时想起 position 和 z-index 的结合使用可生成 层叠上下文 ，大部分同学可能一直认为 z-index 只是定义一个节点在三维空间 Z轴 的层叠顺序，值越高离屏幕观察者就越近。其实这个认识不全面，还需注意以下两点。

- z-index 只在声明定位的节点上起效
- 节点在 Z轴 的层叠顺序依据 z-index 、 层叠上下文 和 层叠等级 共同决定

层叠等级

层叠等级又名**层叠级别**，指节点在三维空间 Z轴 上的上下顺序。在同一 层叠上下文 中，它描述了层叠上下文节点在 Z轴 上的上下顺序；在普通节点中，它描述普通节点在 Z轴 上的上下顺序。

普通节点的 层叠等级 优先由其所在的层叠上下文决定， 层叠等级 的比较只有在当前层叠上下文中才有意义，脱离当前层叠上下文的比较就变得无意义了。

> 成因

很多同学可能认为只有 position 和 z-index 才能让节点生成一个层叠上下文，其实不仅只有这两个属性，还有一些条件也能让节点生成层叠上下文。

- <html> 根结点
- 声明 position:relative/absolute 和 z-index 不为 auto 的节点
- 声明 position:fixed/sticky 的节点
- Flex布局下声明 z-index 不为 auto 的节点

- Grid布局下声明 `z-index` 不为 `auto` 的节点
- 声明 `mask/mask-image/mask-border` 不为 `none` 的节点
- 声明 `filter` 不为 `none` 的节点
- 声明 `mix-blend-mode` 不为 `normal` 的节点
- 声明 `opacity` 不为 `1` 的节点
- 声明 `clip-path` 不为 `none` 的节点
- 声明 `will-change` 不为 `initial` 的节点
- 声明 `perspective` 不为 `none` 的节点
- 声明 `transform` 不为 `none` 的节点
- 声明 `isolation` 为 `isolate` 的节点
- 声明 `-webkit-overflow-scrolling` 为 `touch` 的节点

层叠顺序

层叠顺序指节点发生层叠时按照特定的顺序规则在 `Z轴` 上垂直显示。

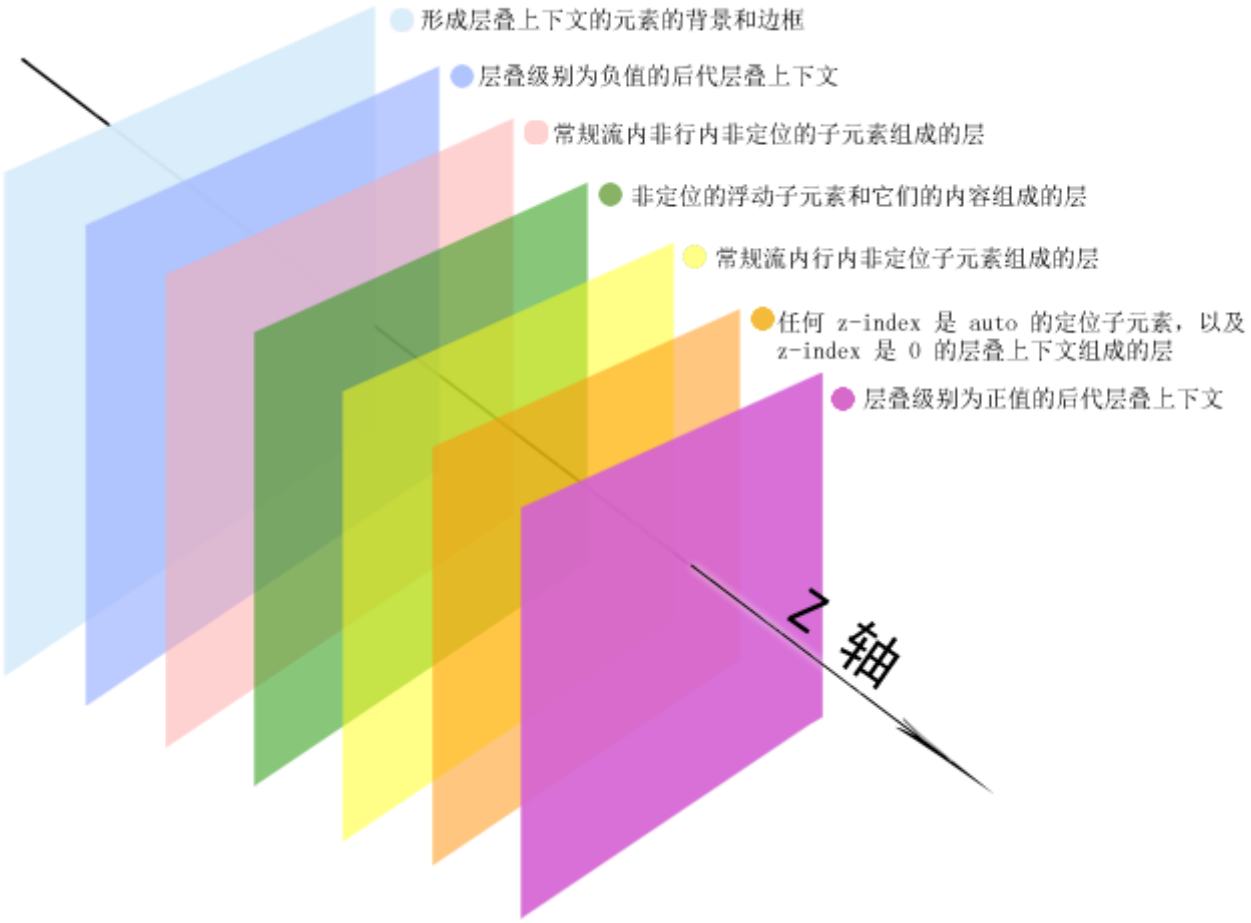
> 脱流元素的层叠顺序

在同一个层叠上下文中，节点会按照 `z-index` 的大小从上到下层叠，若 `z-index` 一致则后面的节点层叠等级要大于前面。脱流元素的层叠顺序就是看 `z-index` 的大小。

> 标准流元素的层叠顺序

标准流元素的层叠顺序稍微有点难记，笔者也未找到特别的记忆方法，只能死记硬背了。以下是层叠顺序从低到高的排列。

- 层叠上下文的 `border` 和 `background`
- `z-index<0` 的子节点
- 标准流内块级非定位的子节点
- 浮动非定位的子节点
- 标准流内行内非定位的子节点
- `z-index:auto/0` 的子节点
- `z-index>0` 的子节点


形成层叠上下文的元素的背景和边框
层叠级别为负值的后代层叠上下文
常规流内非行内非定位的子元素组成的层
非定位的浮动子元素和它们的内容组成的层
常规流内行内非定位子元素组成的层
任何 z-index 是 auto 的定位子元素，以及
z-index 是 0 的层叠上下文组成的层
层叠级别为正值的后代层叠上下文
Z 轴