《CSS世界》读书笔记

第一章第二章可以跳过···（确实比较废话）

1. 流、元素与基本尺寸

3.1 块级元素

1.“块级元素”和“display 为 block 的元素”不是一个概念。例如，<li>元素默认的 display 值是 list-item，<table>元素默认的 display 值是 table，但是它们均是“块级元素”，因为它们都符合块级元素的基本特征，也就是一个水平流上只能单独显示一个元素，多个块级元素则换行显示。

2.由于“块级元素”具有换行特性，因此理论上它都可以配合 clear 属性来清除浮动：

.clear:after {

content: '';

display: table; // 也可以是 block，或者是 list-item

clear: both;

}

实际开发时，我们要么使用 block，要么使用table，并不会使用 list-item的原因：

（1）会出现不需要的项目符号（再加一行 list-style:none 去除即可）

（2）IE 浏览器不支持伪元素的 display 值为 list-item，兼容性不好。对于 IE 浏览器（包括 IE11），普通元素设置 display:list-item 有效，但是 :before/:after 伪元素就不行。

3.1.1 为什么 **list-item** 元素会出现项目符号

1.list-item 元素会出现项目符号是因为生成了一个附加的盒子，学名“标记盒子”（marker box），专门用来放圆点、数字这些项目符号。IE浏览器下伪元素不支持 list-item 或许就是无法创建这个“标记盒子”导致的。

2.每个元素都两个盒子，外在盒子和内在盒子。**外在盒子**负责元素是可以一行显示，还是只能换行显示；**内在盒子**负责宽高、内容呈现什么的（其实叫作“容器盒子”）。

按照 display 的属性值不同，值为 block 的元素的盒子实际由外在的“块级盒子”和内在的“块级容器盒子”组成，值为 inline-block 的元素则由外在的“内联盒子”和内在的“块级容器盒子”组成，值为 inline 的元素则内外均是“内联盒子”。

display属性值是inline-block的元素既能和图文一行显示，又能直接设置 width/height 是因为有两个盒子，外面的盒子是 inline 级别，里面的盒子是 block 级别。

3.1.2 **display:inline-table** 的盒子是怎样组成的

外面是“内联盒子”，里面是“table 盒子”，得到的就是一个可以和文字在一行中显示的表格。

.inline-table {

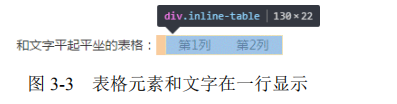
display: inline-table;

width: 128px;

margin-left: 10px;

border: 1px solid #cad5eb;

}



3.1.3 **width/height** 作用在哪个盒子上

内在盒子（“容器盒子”）

3.2 **width/height** 作用的具体细节

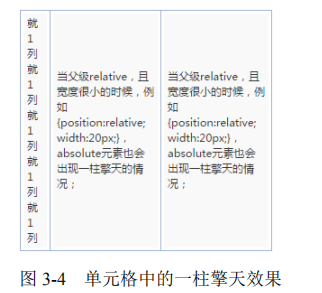
3.2.1 深藏不露的 **width:auto**

1.auto是默认值，至少包含了以下 4 种不同的宽度表现：

（1）充分利用可用空间***（外部尺寸）***。比方说，<div>、<p>这些元素的宽度默认是 100%于父级容器的。这种充分利用可用空间的行为还有个专有名字，叫作 fill-available；

（2）收缩与包裹***（内部尺寸）***。典型代表就是浮动、绝对定位、inline-block 元素或 table 元素，英文称为 shrink-to-fit，直译为“收缩到合适”，我一直把这种现象称为“**包裹性**”。CSS3 中的 fit-content 指的就是这种宽度表现。

（3）收缩到最小***（内部尺寸）***。这个最容易出现在 table-layout 为 auto 的表格中，如图：



当每一列空间都不够的时候，文字能断就断，但中文是随便断的，英文单词不能断。于是，

第一列被无情地每个字都断掉，形成一柱擎天。这种行为在规范中被描述为“preferred minimum width”或者“minimum content width”。后来还有了一个更加好听的名字 min-content。

（4）超出容器限制***（内部尺寸）***。除非有明确的 width 相关设置，否则上面 3 种情况尺寸都不会主动超过父级容器宽度的，但是存在一些特殊情况。例如，内容很长的连续的英文和数字，或者内联元素被设置了 white-space:nowrap，则表现为“恰似一江春水向东流，流到断崖也不回头”。

.father {

width: 150px;

background-color: #cd0000;

white-space: nowrap;

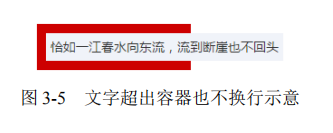
}

.child {

display: inline-block;

background-color: #f0f3f9;

}



子元素既保持了 inline-block 元素的收缩特性，又同时让内容宽度最大，直接无视父级容器的宽度限制。这种现象后来有了专门的属性值描述，这个属性值叫作 max-content。

1. 盒子分“内在盒子”和“外在盒子”，显示也分“内部显示”和“外部显示”，同样地，尺寸也分“内部尺寸”和“外部尺寸”。其中“内部尺寸”英文写作“Intrinsic Sizing”，表示宽度由内部元素决定；还有一类叫作“外部尺寸”，英文写作“Extrinsic Sizing”，宽度由外部元素决定。
2. 外部尺寸与流体特性
3. 正常流宽度

在页面中随便扔一个<div>元素，其尺寸表现就会铺满容器，这就是 block 容器的流特性。

a {

display: block;

width: 100%;

}

表现为“外部尺寸”的块级元素一旦设置了宽度，流动性就丢失了。

流动性，并不是看上去的宽度 100%显示这么简单，而是一种 margin/border/padding 和 content 内容区域自动分配水平空间的机制。

例：上下两个导航均有 margin 和 padding，前者无 width 设置，完全借助流特性，后者宽度 width:100%，结果，后者的尺寸超出了外部的容器，完全就不像“水流” 那样完全利用容器空间，即所谓的“**流动性丢失**”。



1. 格式化宽度

格式化宽度仅出现在“绝对定位模型”中，也就是出现在 position 属性值为 absolute 或 fixed 的元素中。

对于非替换元素，当 left/right 或 top/bottom 对立方位的属性值同时存在的时候，元素的宽度表现为“格式化宽度”，其宽度大小相对于最近的具有定位特性 （position 属性值不是 static）的祖先元素计算。

例：div { position: absolute; left: 20px; right: 20px; }

假设该<div>元素最近的具有定位特性的祖先元素的宽度是 1000 像素，则这个<div>元素的宽度是 960（即 1000−20−20）像素。

和上面的普通流一样，“格式化宽度”具有完全的流体性，也就是 margin、border、

padding 和 content 内容区域同样会自动分配水平（和垂直）空间。

1. 内部尺寸与流体特性

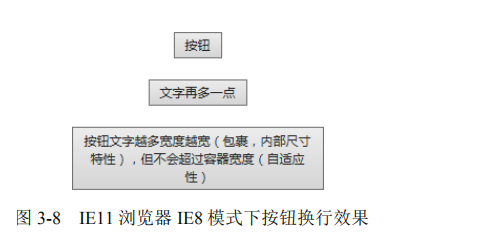
“内部尺寸”，就是元素的尺寸由内部的元素决定，而非由外部的容器决定。如何快速判断一个元素使用的是否为“内部尺寸”：假如这个元素里面没有内容，宽度就是 0，那就是应用的“内部尺寸”。

1. 包裹性

shrink-to-fit，“包裹”+“自适应性”。

“自适应性”，指的是元素尺寸由内部元素决定，但永远小于“包含块”容器的尺寸（除非容器尺寸小于元素的“首选最小宽度”）。

例：按钮就是 CSS 世界中极具代表性的 inline-block 元素，可谓展示“包裹性”最好的例子，具体表现为：按钮文字越多宽度越宽（内部尺寸特性），但如果文字足够多，则会在容器的宽度处自动换行（自适应特性）。



例：页面某个模块的文字内容是动态的，可能是几个字，也可能是一句话。然后，希望文字少的时候居中显示，文字超过一行的时候居左显示。该如何实现？

核心 CSS 代码如下：

.box {

text-align: center;

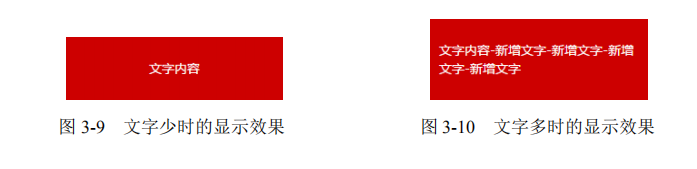
}

.content {

display: inline-block;

text-align: left;

}



除了 inline-block 元素，浮动元素以及绝对定位元素都具有包裹性，均有类似的智能宽度行为。

1. 首选最小宽度

“首选最小宽度”，指的是元素最适合的最小宽度。

假设外部容器宽度是 0，请问里面的 inline-block 元素的宽度是多少？ 是 0 吗？不是。在 CSS 世界中，图片和文字的权重要远大于布局，因此，CSS 的设计者显然是不会让图文在 width:auto 时宽度变成 0 的，此时所表现的宽度就是“首选最小宽度”。

具体表现规则如下：

• 东亚文字（如中文）最小宽度为每个汉字的宽度

• 西方文字最小宽度由特定的连续的英文字符单元决定。并不是所有的英文字符都会组成连续单元，一般会终止于空格（普通空格）、短横线、问号以及其他非英文字符等。例如，“display:inline-block”这几个字符以连接符“-”作为分隔符，形成了“display:inline”和“block”两个连续单元，由于连接符“-”分隔位置在字符后面，因此，最后的宽度就是“display:inline-”的宽度。***如果想让英文字符和中文一样，每一个字符都用最小宽度单元，可以试试使用 CSS 中的 word-break:break-all。***

• 类似图片这样的替换元素的最小宽度就是该元素内容本身的宽度。



例：“首选最小宽度”构建图形，使用一 层 HTML 标签实现“凹”效果（注意要兼容 IE8）

.ao {

display: inline-block;

width: 0;

}

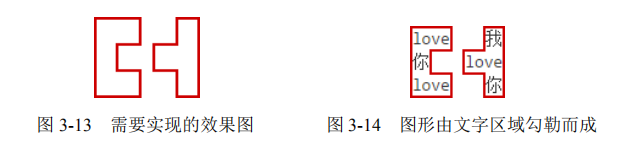
.ao:before {

content: "love 你 love";

outline: 2px solid #cd0000;

color: #fff;

}



1. 最大宽度

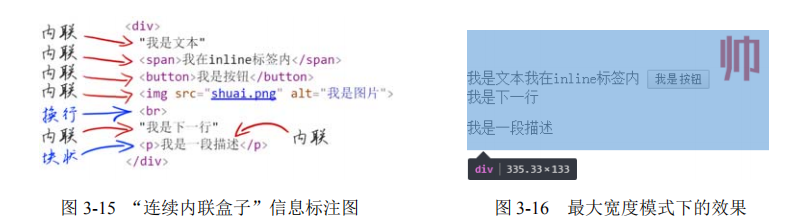
最大宽度就是元素可以有的最大宽度。

如果内部没有块级元素或者块级元素没有设定宽度值，则“最大宽度”实际上是最大的连

续内联盒子的宽度。

“连续内联盒子”指的全部都是内联级别的一个或一堆元素，中间没有任何的换行标签<br>或其他块级元素。

例：



这是一段很平常的 HTML 片段的“连续内联盒子”信息标注图。 其中，有 3 处连续内联盒子，分别是：

• <br>前面的 4 个内联盒子组合；

• <br>后面“我是下一行”字样所在的匿名内联盒子；

• 最后块状<p>标签内的内联盒子，也就是一段文本。

与标注图“内联”文字对应的标注相一致，此时“最大宽度”就是这 3 个连续内联盒子的宽度的最大值。

大部分需要使用“最大宽度”的场景都可以通过设置一个“很大宽度”来实现。注意，这里的“很大宽度”和“最大宽度”是有本质区别的。比方说，有 5 张图片，每张图片宽度 200 像素，假设图片元素紧密排列，则“最大宽度”就是 1000 像素。实际开发的时候，直接设置容器 width:2000px，这里 2000px 就是“很大宽度”，宽度足够大，作用是保证图片不会因为容器宽度不足而不在一行内显示。

只能使用“最大宽度”而不是“很大宽度”的场景：iScroll根据内部元素的尺寸和容器的关系，通过修改内部元素的位置实现滚动效果，可以实现非常平滑的滚动效果。使用 iScroll 模拟水平滚动，只能使用“最大宽度”，滚动到底的时候才是真的到底。

3.2.2 **width** 值作用的细节

width:100px 作用在了 content box 上

有时候，这种宽度设定和表现并不合理。

流动性丢失；与现实世界表现不一致的困扰

3.2.3 CSS 流体布局下的宽度分离原则

“宽度分离原则”，就是 CSS 中的 width 属性不与影响宽度的 padding/border（有

时候包括 margin）属性共存

.father {

width: 180px;

}

.son {

margin: 0 20px;

padding: 20px;

border: 1px solid;

}

3.2.4 改变 **width/height** 作用细节的 **box-sizing**

.box1 { box-sizing: content-box; } /\* 默认值 \*/

.box2 { box-sizing: padding-box; } /\* Firefox 曾经支持 \*/

.box3 { box-sizing: border-box; } /\* 全线支持 \*/

.box4 { box-sizing: margin-box; } /\* 从未支持过 \*/

margin 只有在 width 为 auto 的时候可以改变元素的尺寸

**\*{box-sizing:border-box}不合理之处：**

这种做法易产生没必要的消耗。

search

类型的搜索框，其默认的 box-sizing 就是 border-box（如果浏览器支持），因此，\*对

search 类型的<input>而言也是没有必要的消耗。

这种做法并不能解决所有问题

只有当元素没

有水平 margin 时候，box-sizing 才能真正无计算，而“宽度分离”等策略则可以彻底解决所有的宽度计算的问题。

box-sizing 被发明出来最大的初衷应该是解决替换元素宽度自适应问题。

如textarea<textarea>为替换元素，替换元素的特性之一就是尺寸由

内部元素决定，且无论其 display 属性值是 inline 还是 block。这个特性很有意思，对于

非替换元素，如果其 display 属性值为 block，则会具有流动性，宽度由外部尺寸决定，但

是替换元素的宽度却不受 display 水平影响，因此，我们通过 CSS 修改<textarea>的

display 水平是无法让尺寸 100%自适应父容器的：

textarea {

display: block; /\* 还是原来的尺寸 \*/

}

所以，我们只能通过 width 设定让<textarea>尺寸 100%自适应父容器。那么，问题就来了，

<textarea>是有 border 的，而且需要有一定的 padding 大小，否则输入的时候光标会顶

着边框，体验很不好。于是，width/border 和 padding 注定要共存，同时还要整体宽度 100%

自适应容器。如果不借助其他标签，肯定是无解的

在浏览器还没支持 box-sizing 的年代，我们的做法有点儿类似于“宽度分离”，外面嵌

套<div>标签，模拟 border 和 padding，<textarea>作为子元素，border 和 padding

全部为 0，然后宽度 100%自适应父级<div>。

然而，这种模拟也有局限性，比如无法使用:focus 高亮父级的边框，

因为 CSS 世界中并无父选择器，只能使用更复杂的嵌套加其他 CSS 技巧来

模拟。

textarea {

width: 100%;

-ms-box-sizing: border-box; /\* for IE8 \*/

box-sizing: border-box;

}

合理的css重置：

input, textarea, img, video, object {

box-sizing: border-box;

}

3.2.5 相对简单而单纯的 **height:auto**

height:auto 也有外部尺寸特性,其仅存在于绝对定位模型中，也

就是“格式化高度”。“格式化高度”与“格式化宽度”类似

3.2.6 关于 **height:100%**

height 和 width 还有一个比较明显的区别就是对百分比单位的支持。对于 width 属性，

就算父元素 width 为 auto，其百分比值也是支持的；但是，对于 height 属性，如果父元素height 为 auto，只要子元素在文档流中，其百分比值完全就被忽略了

现对于普通文档流中的元素，百分比高度值要想起作用， 其父级必须有一个可以生效的高度值

为何 **height:100%**无效

有一种看似合理的说法：如果父元素 height:auto 子元素还支持 height:100%，则

父元素的高度很容易陷入死循环，高度无限。例如，一个 <div>元素里面有一张

vertical-align 为 bottom 同时高度为 192 像素的图片，此时，该<div>高度就是 192

像素，假设此时插入一个子元素，高度设为 100%，如果此时 height:100%可以计算，则

子元素应该也是 192 像素。但是，父元素 height 值是 auto，岂不是现在高度要从原来的

192 像素变成 384 像素，然后 height:100%的子元素高度又要变成 384 像素，父元素高度

又双倍……死循环了！

实际上，这种解释是错误的

证据就是宽度也存在类似场景，但并没

有死循环。例如，在下面这个例子中，父元素采用“最大宽度”，然后有一个 inline-block

子元素宽度 100%：

<div class="box">

<img src="1.jpg">

<span class="text">红色背景是父级</span>

</div>

.box {

display: inline-block;

white-space: nowrap;

background-color: #cd0000;

}

.text {

display: inline-block;

width: 100%;

background-color: #34538b;

color: #fff;

}

如果按照上面“高度死循环”的解释，这里也应该“宽度死循环”，因为后面的 inline-block

元素按照我们的理解应该会让父元素的宽度进一步变大。但实际上并没有，宽度范围可能超出

你的预期（见图 3-24）。父元素的宽度就是图片加文字内容的宽度之和。



为什么会这样表现呢？

要明白其中的原因要先了解浏览器渲染的基本原理。首先，先下载文档内容，加载头部的

样式资源（如果有的话），然后按照从上而下、自外而内的顺序渲染 DOM 内容。套用本例就是，

先渲染父元素，后渲染子元素，是有先后顺序的。因此，当渲染到父元素的时候，子元素的

width:100%并没有渲染，宽度就是图片加文字内容的宽度；等渲染到文字这个子元素的时候，

父元素宽度已经固定，此时的 width:100%就是已经固定好的父元素的宽度。宽度不够怎么

办？溢出就好了，

如果 height 支持任意元素 100%，也是不会死循环的。和宽度类似，静态

渲染，一次到位。

为何宽度支持，高度就不支持呢？规范中其实给出了答案。如果包含

块的高度没有显式指定（即高度由内容决定），并且该元素不是绝对定位，则计算值为

auto。一句话总结就是：因为解释成了 auto。要知道，auto 和百分比计算，肯定是算

不了的：

'auto' \* 100/100 = NaN

宽度的解释却是：如果包含块的宽度取决于该元素的宽度，那么产生的布局在 CSS 2.1

中是未定义的,就按照包含块真实的计算值作为 百分比计算的基数

如何让元素支持 **height:100%**效果

设定显式的高度值

例如，设置 height:600px，或者可以生效

的百分比值高度。例如，我们比较常见的：

html, body {

height: 100%;

}

使用绝对定位。

例如：

div {

height: 100%;

position: absolute;

}

此时的 height:100%就会有计算值，即使祖先元素的 height 计算为 auto 也是如此。

需要注意的是，绝对定位元素的百分比计算和非绝对定位元素的百分比计算是有区别的，区别

在于绝对定位的宽高百分比计算是相对于 padding box 的，也就是说会把 padding 大小值计算

在内，但是，非绝对定位元素则是相对于 content box 计算的。

3.3 CSS **min-width/max-width**和**min-height/max-height** 二三事

3.3.1 为流体而生的 **min-width/max-width**

一种特

定区间内的自适应布局方案就诞生了，网页宽度在 1200～1400 像素自适应，既满足大屏的大气，

又满足笔记本的良好显示，此时，min-width/max-width 就可以大显神威了：

.container {

min-width: 1200px;

max-width: 1400px;

}

在公众号的热门文章中，经常会有图片，这些图片都是用户上传产生的，因此尺寸会有大

有小，为了避免图片在移动端展示过大影响体验，常常会有下面的 max-width 限制：

img {

max-width: 100%;

height: auto!important;

}

height:auto 是必需的，否则，如果原始图片有设定 height，max-width 生效的时候，

图片就会被水平压缩。强制 height 为 auto 可以确保宽度不超出的同时使图片保持原来的比

例。但这样也会有体验上的问题，那就是在加载时图片占据高度会从 0 变成计算高度，图文会

有明显的瀑布式下落。

3.3.2 与众不同的初始值

min-width/min-height 的初始值是 auto，max-width/max

height的初始值是none

3.3.3 超越**!important**，超越最大

超越!important 指的是 max-width 会覆盖 width，会超越!important 的权重

超越最大指的是min-width覆盖max-width，此规则发生在min-width和max-width

冲突的时候。

3.3.4 任意高度元素的展开收起动画技术

.element {

max-height: 0;

overflow: hidden;

transition: max-height .25s;

}

.element.active {

max-height: 666px; /\* 一个足够大的最大高度值 \*/

}

，如果 max-height 值太大，在收起的时候可能会有“效果延迟”的问

题，

max-height 使用足够安全的最小值，这样，收起时即使有延迟，也

会因为时间很短，很难被用户察觉，并不会影响体验

3.4 内联元素

3.4.1 哪些元素是内联元素

内联元素”的“内联”特指“外在盒子”，和“display 为 inline

的元素”不是一个概念！inline-block 和 inline-table 都是“内联元素”，因为它们的

“外在盒子”都是内联盒子。自然 display:inline 的元素也是“内联元素”，那么，<button>

按钮元素是内联元素，因为其 display 默认值是 inline-block；<img>图片元素也是内联

元素，因为其 display 默认值是 inline 等。

3.4.2 内联世界深入的基础—内联盒模型

（1）内容区域（content area）。内容区域指一种围绕文字看不见的盒子，其大小仅受字符本身

特性控制，本质上是一个字符盒子（character box）；但是有些元素，如图片这样的替换元素，其内

容显然不是文字，不存在字符盒子之类的，因此，对于这些元素，内容区域可以看成元素自身。



（2）内联盒子（inline box）。“内联盒子”不会让内容成块显示，而是排成一行，这里的“内

联盒子”实际指的就是元素的“外在盒子”，用来决定元素是内联还是块级。该盒子又可以细分

为“内联盒子”和“匿名内联盒子”两类：



如果外部含内联标签（<span>、<a>和<em>等），则属于“内联盒子”（实线框标注）；如

果是个光秃秃的文字，则属于“匿名内联盒子”（虚线框标注）。

需要注意的是，并不是所有光秃秃的文字都是“匿名内联盒子”，其还有可能是“匿名块

级盒子”，关键要看前后的标签是内联还是块级。

（3）行框盒子（line box）。例如：



每一行就是一个“行框盒子”（实线框标注），每个“行框盒子”又是由一个一个“内联盒子”

组成的。

（4）包含盒子（containing box）（包含块）。例如：



<p>标签就是一个“包含盒子”（实线框标注），此盒子由一行一行的“行框盒子”组成。

3.4.3 幽灵空白节点

以举一个最简单的例子证明“幽灵空白节点”确实存在， CSS 和 HTML 代码如下：

div {

background-color: #cd0000;

}

span {

display: inline-block;

}

<div><span></span></div>

结果，此<div>的高度并不是 0，而是如图 3-29 所示有高度。

这着实很奇怪，内部的<span>元素的宽高明明都是 0，标

签之间也没有换行符之类的嫌疑，怎么<div>的高度会是图

3-29 中所示的 18 像素呢？

作祟的就是这里的“幽灵空白节点”，如果我们认为在

<span>元素的前面还有一个宽度为 0 的空白字符，是不是一

切就解释得通呢？



“幽灵空白节点”

实际上也是一个盒子，不过是个假想盒，名叫“strut”，中文直译为“支柱”，是一个存在于每个“行

框盒子”前面，同时具有该元素的字体和行高属性的 0 宽度的内联盒。