《CSS世界阅读》笔记

> 祝梦想成真！ —— 张鑫旭

流、元素与基本尺寸

块级元素

1. 常见的块级元素有`<div>`、`<li>`、`<table>`等。
2. 需要注意的是 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是list-item，`<table>`元素默认的display是table，但是它们均属于块级元素，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，多个块级元素则会换行。
3. 正由于块级元素的换行特性，因此可以配合clear来清除浮动。

display-block

1. 穿着inline的皮藏着block的心。
2. 每个元素都有2个盒子（***容器盒子***），外在盒子和内在盒子，外在盒子负责元素可以是一行显示还是换行显示。内在盒子负责宽高、内容呈现。
3. 于是按照display属性值的不同，值为block的元素则外在由‘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的元素内外均是‘内联盒子’。
4. 以此理解，`display:block`可以脑补成`display:block-block`。

width/height 作用在哪个盒子上，作用细节

作用在内在盒子上，也就是盒子容器

深藏不露的width：auto

width的默认值为auto，auto包含了4种不同的宽度表现

1. 充分利用可用空间
    - `<div>`、`<p>`这些元素宽度是100%于父级元素（fill-available）

2. 收缩和包裹
    - 典型代表就是浮动、绝对定位、inline-block元素、table元素。会使元素收缩至内容宽度。这种现象称为包裹性。
3. 收缩到最小
    - 最常出现在table布局为auto的表格中。例子
    - 当每一列空间都不够的时候，文字能断就断。中文是随便断的，英文单词不能断，于是例子中的第一列被断掉了，形成一柱擎天。称为min-content
4. 超出容器限制
    - 除非有明确的width设置，否则以上三种情况尺寸都不会主动超过父级容器宽度。但存在一些特殊情况。 比如很长的连续英文和数字，或者内联元素被设置为 `white-space:nowrap`（不换行）

外部尺寸与流体特性

1. 正常流宽度。当我们在一个容器里倒入足量的水，水一定均匀的铺满这个容器。这就是block容器的特征，在所有浏览器中都表现一致。
    - 块级元素一旦设置了宽度，流动性就丢失了，失去了各内容区域自动分配水平空间的机制。
    - demo 加了 `width:100%`，导致边框超出容器。
    - 无宽度准则，借助浏览器原生流的特性，外部容器尺寸变化，内部元素也可以自适应。

> 鑫三无准则（无宽度，无图片，无浮动）

2. 格式化宽度。仅出现在绝对定位模型中。也就是 `position:absolute` 或 `fixed` 元素中。默认情况下，绝对定位元素的宽度具有包裹性，宽度由内部尺寸决定。但有一种情况例外。
    - 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’，其宽度相对于具有定位特性的祖先元素计算。
    - 格式化宽度 具有完全的流动性，margin、border、padding、content内容区域同样会自动分配。

```
<div class="a">
  <div class="b">
  </div>
</div>
```

```
html,body {
  margin:0;
  height: 100%;
}
.a {
 position: relative;
  background-color: red;
  width: 100%;
  height: 100%;
}
.b {
  position: fixed;
  top: 50px;
  bottom: 50px;
  background-color: green;
}
```

内部尺寸与流体特性

所谓 内部元素，就是元素尺寸由内部元素决定。

1. 包裹性
   - button是css世界中极具代表性的inline-block元素。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，则会在容器宽度处换行。（自适应特征）。例如容器宽度240px，button会在240px处换行。
   - `<button>`标签按钮才会自动换行,`<input>`标签按钮，默认`white-space:pre`，是不会换行的，需要设置为normal。
   - demo

```
可以实现文字少的时候居中显示，多文字的时候居左显示。
  .box {
      text-align:center;
  }
  .content {
      display:inline-block;
       text-align:left;
  }
```

2. 首选最小宽度
   - 当外部容器的宽度为0时，里面的inline-block元素的宽度并不是0。
   - 东亚文字最小宽度为每个字的宽度。

- 西方文字最小宽度为特定的连续英文字符单元决定。
- 可以使用`word-break:break-all`使英文单词与汉字一样表现为最小宽度。

3. 最大宽度
   - 如果容器内部没有块级元素或者块级元素没有设定宽度，则最大宽度实际上是最大连续内联盒子的宽度。
   - demo

width值得作用细节

要知道width属性具体数值的表现，就需要了解css世界中与尺寸相关的重要概念***盒尺寸***css盒模型结构分为 content、padding、border、margin。我们的这个‘内在盒子’被分为4个盒子，content box、padding box、border box、margin box。但唯独margin box 没有对应的css关键字名称，因为目前没有任何场景需要用到margin box。

width：100px是如何作用到`<div>`元素上去的

在css2.1规范中，content box环绕着width和height给定的矩形。width:100px作用在content box上，由于`<div>`元素默认的padding、border、margin都是0，因此该元素的宽度就是100px。

对于块级元素，如果width:auto,则元素会如水流般充满整个容器。而一旦设置了width为具体数值，则元素的流动性就会被阻断。尤其宽度作用在content box上，更是内外流动性全无。因此，**无宽度准则**更灵活，容错性更强。

css流体布局下的宽度分离原则

所谓‘宽度分离原则’，就是css中width属性不与影响宽度的padding/border（有时候包括margin）属性共存。也就是不能出现一下情况

```
.box {
    width:100px;
    border:1px solid;
}
.box {
    width:100px;
    padding:10px;
}
```

‘宽度分离原则’采用width独立占用一层标签，padding、border、margin利用流动性在内部自适应呈现。

```
.father {
    width:100px;
}
.son {
    margin:0 auto;
    padding:20px;
    border:1px solid;
}
```

使用宽度分离，可以使样式更好维护，让流动性动态去计算尺寸，省去人为计算。

改变width/height 作用细节的box-sizing

1. 虽然box-sizing被直译为‘盒尺寸’。实际上，其更准确的叫法应该是‘盒尺寸的作用细节’，更通俗点叫‘width的作用细节’。也就是说box-sizing属性的作用是改变width的作用细节。
2. 默认情况下，width是作用在content box 上。所谓box-sizing:border-box，就是让宽度作用在border-box上，此时 content box 就从宽度值中释放，形成局部流动性，和padding值一起自动分配width值。
3. 为什么box-sizing不支持margin-box
    - 不支持margin-box的最大原因是因为它本身就没有价值。因为本身并不会改变元素尺寸的盒子。
    - 另一个原因牵扯到语义。如果box-sizing开了先河支持了margin-box，margin box就变成了一个‘显示的盒子’。background-origin等属性何去何从，支持还是不支持。css规范中写明了 margin的背景永远是透明的。
    - 使用场景上，对于box-sizing的margin-box效果，如果是IE10及以上版本浏览器，可以试试flex布局。

如何评价 `*{box-sizing:border-box}`

1. 这种做法易产生没必要的消耗。通配符`*`应该是一个慎用的选择器。因为它会选择所以得标签元素，对应普通内联元素，box-sizing无论是什么值，对于渲染表现都没有影响。因此，`*`对于这些元素是没必要的消耗。
2. 这种做法并不能解决所有问题。

为何height：100%无效

规范中给出了答案。如果包含块的高度没有显示指定(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auto，auto和百分百计算是计算不了的。

如何让元素支持hegith:100%

1. 设定显式的高度值
2. 使用绝对定位`div {height:100%;position:absolute;}`此时height就会有计算值。即使祖先元素计算为auto也是如此。需要注意的是，绝对定位元素的百分百计算和非绝对定位元素的百分百计算是有区别的。区别在于绝对定位元素的宽高百分百计算是相对于padding box的，也就是说会把padding的大小值计算在内。但是，非绝对定位元素则是相对于content box的。 例子

超越！important，超越最大

css世界中，min/max width和 min/max height属性间，以及与width、height之前有一套互相覆盖的规则。这套规则用一句比较通俗的话概况就是：超越！important，超越最大。

1. 超越！important指的是max-height会覆盖height，而且这种覆盖不是普通的覆盖。是超越覆盖。

```
<img src="a.png" style="width:100px !important;">
<style>
 img {
     max-width:50px;
 }
</style>
```

2.超越最大。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。例如

```
 .content {
     min-width:1400px;
     max-width:1200px;
 }
```

最小宽度比最大宽度还要大的时候，产生冲突。遵循超越最大规则。（不是后来居上规则），min-width活下来，max-height被忽略。

任意高度的元素的展开收起动画技术

```
.element {
    max-height:0;
    overflow:hidden;
    transition:max-height 0.25s;
}
.element.acitve {
    max-height:666px; /*一个足够大的高度*/
}
```

内联元素

1. 从定义看，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。`<button>`按钮元素也是内联元素，因为其display默认值是inline-block。`<img>`图片元素也是内联元素，因为display默认值是inline。
2. 从表现看，内联元素的典型特征就是可以和文字在一行显示。文字是内联元素，图片是内联元素，按钮是内联元素。输入框、下拉框等原生控件也是内联元素。

内联盒模型

```
<p>这是一行普通的文字，这个有个<em>em</em>标签</p>
```

1. 内容区域，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。本质上是个字符盒子。我们可以把文本选中的背景色区域作为内容区域。
2. 内联盒子。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。该盒子又可以细分成 内联盒子 和 匿名内联盒子。如果外部含内联标签（span、a、em等），则属于内联盒子。如果是光秃秃的文字，则属于匿名内联盒子。
3. 行框盒子。每一行就是一个行框盒子。内个行框盒子又是一个个内联盒子组成

的。

4. 包含盒子。p标签就是一个包含盒子，此盒子由一行行行框盒子组成。

盒尺寸四大家族

content 与替换元素

什么是替换元素

替换元素就是内容可以被替换。比如

```
<img src="1.jpg">
```

如果我们把src替换为别的路径，内容也可以被替换。这种通过修改某个属性值呈现内容就可以被替换的元素就称为 替换元素。因此

```
<img>、<object>、 <video>、<iframe>、<textare>、<input>
```

都是典型的替换元素。替换元素除了内容可替换这一特性外，还有以下特性。

1. 内容外观不受页面css影响。用专业的话讲就是在样式表现在css作用域外。如何更改替换元素本身的外观。需要类似appearance属性。或者浏览器自身暴露的一些样式接口。例如::-ms-check {}可以更改高版本IE浏览器下单复选框的内间距，背景色等样式。
2. 有自己的尺寸。在Web中，很多替换元素在没有明确尺寸设定的情况下，其默认的尺寸是300像素x150像素。如 `<video><iframe>` 或者 `<canvas>` 等。也少有部分替换元素为0像素的，如 `<img>` ，而表单元素的替换元素尺寸则和浏览器有关，没有明显的规律。
3. 在很多css属性上有自己的一套表现规则。比较有代表性的就是vertical-align属性。对于替换元素和非替换元素，vertical-align属性值的解释是不一样的。比方说vertical-align的默认值是baseline。基线对齐，被定义为字符x的下边缘。但在替换元素中，往往没有字符x，于是替换元素的基线就被硬生生定义为元素的下边缘。

替换元素的尺寸计算规则

1. 固有尺寸指的是替换内容原本的尺寸。例如，图片，视频作为一个独立文件存在的时候，都是有着自己的宽度和高度的。这个宽高就是固有尺寸。对于表单类替换元素，固有尺寸可以理解成 不加修饰的默认尺寸。
2. HTML尺寸。 HTML尺寸只能通过HTML原生属性改变。这些HTML属性包括`<img>`的width和height，`<input>`的size属性，`<textarea>`的cols和rows等。

```
<img width="300" height="300">
<input type="file" size="30">
<textarea cols="20" rows="5"><textara>
```

3. css尺寸特指可以通过css的width和height或者max-width、min-width和max-height和min-height设置的尺寸。对应盒尺寸的content box。

可以影响替换元素尺寸的三层结构

- 如果没有css尺寸和HTML尺寸，则使用固有尺寸作为最终的宽高。
- 如果没有css尺寸，则使用HTML尺寸作为最终宽高。
- 如果有css尺寸，则最终尺寸由css属性决定。如`<img src="1.jpg" width="100" height="100"> img {width:200px;}`
- 如果固有尺寸含有固有的宽高比例，同时仅设置了宽度或仅设置了高度，则元素按照固有的宽高比例显示。如`<img src="1.jpg"> img {width:200px;}`
- 如果上面的条件都不符合，则最终宽度表现为300像素。如`<video></video>`

温和的padding属性

padding与元素尺寸

因为css中默认的box-sizing是content-box。所以使用padding会增加元素的尺寸。新人难免会在padding的尺寸问题上拆坑。很多人设置box-sizing为border-box，甚至全局的box-sizing。（具体参考前面的回答）

标签元素内置的padding

1. ol/ul 列表内置padding-left。但单位是px不是em。Chrome下是40px，由于使用的是px这个绝对单位，因此，如果font-size很小的时候，列表的项目符号就会距离左边缘很开，font-size比较大的时候，则项目符号可能跑到`<ul>/<ol>`外面。根据经验，当font-size时12px至14px时，22px是比较好

的一个padding-left。

2. 很多表单元素都内置padding
   - 所以浏览器 `<input><textarea>` 输入框内置padding
   - 所以浏览器 `<button>` 按钮内置padding
   - 部分浏览器 `<select>` 下拉框内置padding
   - 所以浏览器 `<radio> <checkbox>` 无内置padding
   - `<button>` 按钮元素的padding最难控制。

padding与图形绘制

用一个元素模拟双层圆点。

```
.icon {
    display: inline-block;
    width:100px;
    height:100px;
    padding:10px;
    border:10px solid;
    border-radius:50%;
    background-color:red;
    background-clip:content-box;
}
```

激进的margin属性

margin与元素尺寸以及相关布局

margin同样可以改变元素的可视尺寸，但是和padding几乎是互补态势。对于padding，元素设定了width'值或者保持包裹性的时候，会改变元素可视尺寸。但是，对于margin则相反，元素设定了width值或者保持包裹性的时候，margin对尺寸没有影响，只是元素充分利用可用空间状态的时候，margin才可以改变元素的可视尺寸。比如：

```
.father {
    width:300px;
    margin:0 -20px;
}
```

此时元素宽度还是300px，尺寸无改变。因为只要宽度设定，margin就无法改变元素尺寸。这和pading是不一样的。但是：

```
<div class="father">
  <div class="son"></div>
</div>

<style>
 .father {
     width:300px;
 }
 .son {
     margin:0 -20px;
 }
</style>
```

son元素宽度就是340px，尺寸通过负值设置变大了，因为此时宽度表现是充分利用可用空间。只要元素的尺寸表现符合 充分利用可用空间。无论是垂直方向还是水平方向，都可以通过margin改变尺寸。css世界默认的流方向是水平方向。因此，对于普通流体元素，margin只能改变元素水平方向尺寸。对视对于具有拉伸特性的绝对定位元素，则水平或者垂直方向都可以。

正确看待css世界里的margin合并

什么是margin合并

块级元素的上外边距与下外边距有时会合并为单个外边距。这种现象被称为margin合并。从此定义上，我们可以捕捉两点重要的信息。

1. 块级元素。但不包含浮动和绝对定位元素。
2. 只发生在垂直方向。由于默认文档流是水平流，因此发生margin合并就是垂直方向。

margin合并的3种场景

1. 相邻兄弟元素margin合并。这是margin合并最常见、最基本的

```
p {margin:1em 0}
<p>第一行</p>
<p>第二行</p>
```

2. 父级和第一个/最后一个子元素。demo

3. 空块级元素的margin合并。

```
.father {
    overflow:hidden;
}
.son {
    margin: 1em 0;
}
<div class="father">
  <div class="son"></div>
</div>
```

此时father元素div元素的高度仅仅为1em，因为son这个空div元素的margin-top和margin-bottom合并在一起了。这也是上一节margin：50%最终宽高比是2：1的原因。因为垂直方向的上下margin值合二为一，所以垂直方向的外部尺寸只有水平方向的一半。

margin合并的计算规则

margin合并的规则总结为 **正正取大值，正负值相加，负负最负值**

margin合并的意义

对于兄弟元素margin合并的作用和em类似。都是让图文信息的排版更加舒服自然。假如没有margin合并的说法，那么连续段落或者列表之类的尾项间距会和其他兄弟标签成1：2关系。文章标题距离顶部会很近，而和虾米的文章相亲内容距离又很开，造成内容上下间距不一致的情况。

对于父子margin合并的意义在于：在页面中任何地方嵌套或者直接放入任何裸div，都不会影响原来的块级布局。

自身margin合并的意义在于：可以避免不小心遗落或者生成空标签影响排版和布局。

魔鬼属性float

float的本质与特性

浮动的本质就是为了实现文字的环绕效果。而这种文字环绕，主要指的是文字环绕图片显示的效果。从float属性的设计初衷来看，当下那些漫天飞舞的浮动属性完全就是滥用了。代码实现上就是把元素一个个定宽定高，通过浮动一个个堆积起来。

理论上几乎可以把整个页面结构弄出来，而且内联元素的间隔问题，margin合并问题都没有。乍一看，float好像能满足我们布局的需求，但实际上，这种砌砖头的布局缺少弹性，一旦某个元素的高度变化，则一面列表可能会发生布局错位，或是我们要调整某个元素的宽度，则牵一发而动全身。

鑫三无准则之无浮动

之所以要无浮动。是因为纯浮动布局容错性差，容易出现比较严重的布局问题，另一个原因是float本身就是魔鬼属性，容易出现意料之外的情况，除了float属性本身的特性呆滞布局问题外，还包括诸多兼容性问题。float存在的特性：

1. 包裹性
2. 块级化并格式化上下文(元素一旦float不为none，则display计算值就是block或者table，除了inline-table计算为table外)
3. 破坏文档流
4. 没有任何margin合并

float的作用机制

float属性有个著名的表现特性就是会让父元素高度塌陷。在大多场景下，这种特性会影响正常的布局。float属性原本作用就是为了实现文字的环绕效果，设计者为了在古老的盒模型中实现文字环绕效果，想到了破坏文档流这一招。图文展示只是web展示的一小部分，而文字环绕效果现在已经不流行了，于是float很少发挥其原本的作用，反而被大肆使用满屏布局。显然，布局的时候是不不需要父元素高度塌陷的，于是高度塌陷这种特性反而成为float属性的坑。

float克星clear

css有个专门处理float属性带来高度塌陷的属性clear。我们平时除了clear:both这个声明比较大之外，left和right这2个属性几乎无人问津。因为确实没什么用，凡是clear:left或者clear:right起作用的地方，一定可以使用both替换。

成事不足败事有余的clear

clear属性只有块级元素才有效。而::after等伪元素默认是内联元素。这就是借助伪元素清除浮动影响时需要设置display属性值得原因。由于clear:both的作用本质是让自己不和float元素在一行显示，并不是真正意义上的清除浮动。因此，flaot元素一些不好的特性依然存在。于是会出现

1. 如果clear:both元素前面的元素就是float元素，则margin-top负值即使设置成-9999px，也不见任何效果。

2. clear:both后面的元素依旧可能发生文字环绕现象。

```
<div class="father">
  <img
src="https://ss3.baidu.com/9fo3dSag_xI4khGko9WTAnF6hhy/image/h%3
D300/sign=6d0bf83bda00baa1a52c41bb7711b9b1/0b55b319ebc4b745b19f8
2c1c4fc1e178b8215d9.jpg" alt="">
   father元素内部的文字
</div>
<div>father元素后面的文字会出现文字环绕效果</div>

<style>
  .father::after {
    content:'';
    display:table;
    clear:both;
  }
  .father img {
    float:left;
    width:100px;
    height:100px;
  }
  .father + div {
    margin-top:-2px;
  }
</style>
```

由此可见，clear:both只能在一定程度上消除浮动的影响，要想完美的去处浮动元的影响，还需要其他css声明。

css世界的结界--BFC

BFC的定义

BFC全称为 black farmatting context，中文叫 块级格式化上下文。可以用 css世界的结界 来概括BFC的特性。结界 这个词大家都应该理解，指通过一些特定的手段形成的封闭空间，里面的人出不去，外面的人进不来，具有极强的防御力。BFC的特性表现如出一辙。

大家记住下面这个表现原则。如果一个元素具有BFC，内部子元素再怎么翻江倒海、翻云覆雨都不会影响外部的元素。所以，BFC是不可能发生margin重叠的，因为margin重叠是会影响外面元素的元素的。BFC元素可以用来清除浮动的影响。那什么时候触发BFC呢

1. `<html>`根元素
2. float的值不为none
3. overflow的值为auto、scroll、hidden
4. display的值为table-cell、table-caption和inline-block中的任何一个
5. positon的值为relative和static

换言之，只要元素符合上面任意一个条件，就无须使用clear:both属性去清除浮动。

最佳结界 overflow

要想彻底清除浮动的影响，最适合的属性不是clear而是overflow。一般使用overflow:hidden,利用BFC的结界特性彻底解决浮动对外部或者兄弟元素的影响。虽然有其他css声明也能清除浮动，但基本上都会让元素的宽度表现为包裹性，也就是会影响原来的样式布局。

依赖overflow的样式表现

单行文字溢出点点效果

```
.ell {
    text-overflow:ellipsis;
    white-space:nowrap;
    overflow:hidden;
}
```

目前对-webkit-私有前缀支持良好的浏览器还可以实现多行文字打点效果，但是无须依赖overflow:hidden。比如说，最多显示2行文字内容

```
.ell-rows-2 {
    display:-webkit-box;
    -webkit-box-orient:vertical;
    -webkit-line-clamp:2;
}
```

float的兄弟position:absolute

我一直认为position:absolute和float:left/right 是兄弟元素。都具备块级化、包裹

性、破坏性等特性，不少布局场合甚至可以相互替代。例如，块级化和浮动类似，元素一旦position属性值为absolute或fixed，其display计算值就是block或table，例如 `<span>` 元素默认是inline，但是一旦设置为position：absolute，其display计算值就变成了block。又比如 破坏性。指的是破坏正常流的特性，和float类似，虽然absolute破坏正常流来实现自己的特性表现，但本身还是受普通的流体元素布局、位置甚至内联相关属性的影响。又比如两者都能块级格式化上下文，也就是BFC。又比如说两者都具有包裹性，也就是 尺寸收缩包裹，同时具有自适应性。

absolute的包含块

包含块这个概念大家一直都有接触过，就是元素用来计算和定位的一个框。对于这些计算规则，规范是有明确定义的。具体如下：

1. 根元素(很多场景下可以看成是 `<html>`)被称为 初始包含块，其尺寸等同于浏览器可视窗口大小
2. 对于其他元素，如果该元素的position是relative或static，则包含块由其最近的块容器祖先∑content box形成
3. 如果元素position是fixed，则包含块是初始包含块
4. 如果元素position是absolute，则包含块由最近的position不为static的祖先元素建立。

强悍的position：fixed固定定位

position：fixed不一样的包含块

position：fixed的包含块是根元素，我们可以将其近似看做 `<html>` 元素。换句话说，唯一可以限制固定定位元素的就是 `<html>` 根元素，而根元素就那么一个。

position：fixed与背景锁定

蒙层弹窗是网页常见的交互，其中黑色半透明全屏覆盖的蒙层基本上都是使用position:fixed定位实现的。但是，如果细致一点就会发现蒙层无法覆盖浏览器右侧的滚动栏，并且鼠标滚动的时候后面的背景内容依然可以被滚动，并没有锁定，体验略打折。如果希望背景锁定，如何实现?

要想解决一个问题，可以从发生这个问题的原因入手。position：fixed蒙层之所以能滚动，是因为滚动元素是根元素，正好是position：fixed的包含块。所以希望背景能锁定，可以借鉴absolute模拟fixed定位的思路。让页面滚动条由内部的普通元素产生。如果网页滚动结构不方便调整，则需要借助js来实现锁定。如果是移动端项目，组织touchmove事件的默认行为可以防止滚动。如果是桌面端项目，可以让根元素overflow：hidden。但是Windows操作系统下的浏览器滚动条都是占据一定

宽度的，滚动条的消失必然导致页面的可用宽度变化，页面会产生体验更糟糕的晃动。我们需要找个东西填补消失的滚动条就好。这时候就用到border

```
var widthBar = 17,root = document.documentElement
if(typeof window.innerWidth == 'number'){
    widthBar = window.innerWidth - root.clientWidth
}
root.style.overflow = 'hidden'
root.style.borderRight = widthBar + 'px solid transparent'
```