谈谈一些有趣的CSS题目（11~15） #5

New issue

Open chokcoco opened this issue on 7 Nov 2016 · 8 comments

chokcoco commented on 7 Nov 2016 • edited Owner

11、IFC、BFC、GFC 与 FFC 知多少

这么多 `*FC` 都是些啥？

FC 即是 Formatting Contexts ，译作格式化上下文。 `*FC` 可以称作视觉格式化模型。CSS 视觉格式化模型 (visual formatting model)是用来处理文档并将它显示在视觉媒体上的机制。这是 CSS 的一个基础概念。

比较常见的是 CSS2.1 规范中的 IFC（Inline Formatting Contexts）与 BFC（Block Formatting Contexts），至于后面两个，则是 CSS3 新增规范，GFC（GridLayout Formatting Contexts）以及 FFC（Flex Formatting Context）。

FC 是网页CSS视觉渲染的一部分，用于决定盒子模型的布局、其子元素将如何定位以及和其他元素的关系和相互作用。

理解各种 FC 背后的原理是掌握各类 CSS 布局的关键。

先了解几个概念，

Box

Box 是 CSS 布局的对象和基本单位，直观点说就是一个页面是由很多个 Box (即boxes)组成的，元素的类型和 display 属性决定了 Box 的类型。

1. block-level Box：当元素的 CSS 属性 display 为 block, list-item 或 table 时，它是块级元素 block-level 。块级元素（比如 `<p>`)视觉上呈现为块，竖直排列。
   每个块级元素至少生成一个块级盒（block-level Box）参与 BFC ，称为主要块级盒(principal block-level box)。一些元素，比如 `<li>` ，生成额外的盒来放置项目符号，不过多数元素只生成一个主要块级盒。

2. Inline-level Box：当元素的 CSS 属性 display 的计算值为 inline, inline-block 或 inline-table 时，称它为行内级元素。视觉上它将内容与其它行内级元素排列为多行。典型的如段落内容，有文本或图片，都是行内级元素。
   行内级元素生成行内级盒(inline-level boxes)，参与行内格式化上下文 IFC 。

3. flex container：当元素的 CSS 属性 display 的计算值为 flex 或 inline-flex ，称它为弹性容器。
   `display:flex` 这个值会导致一个元素生成一个块级（block-level）弹性容器框。
   `display:inline-flex` 这个值会导致一个元素生成一个行内级（inline-level）弹性容器框。

4. grid container：当元素的 CSS 属性 display 的计算值为 grid 或 inline-grid，称它为栅格容器。

> 栅格盒模型值，是一个仍处于实验中的属性。

块容器盒（block container box）

只包含其它块级盒，或生成一个行内格式化上下文(inline formatting context)，只包含行内盒的叫做**块容器盒子。**

也就是说，块容器盒要么只包含行内级盒，要么只包含块级盒。

> 块级盒（block-level Box）是描述元素跟它的父元素与兄弟元素之间的表现。

> 块容器盒（block container box）描述元素跟它的后代之间的影响。

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

6 participants

块盒（BLock Boxes）

同时是块容器盒的块级盒称为块盒(block boxes)

> 顶住，概念真的很多。。。

行盒(Line boxes)

行盒由行内格式化上下文(inline formatting context)产生的盒，用于表示一行。在块盒里面，行盒从块盒一边排版到另一边。 当有浮动时, 行盒从左浮动的最右边排版到右浮动的最左边。

IFC（Inline Formatting Contexts）行内级格式化上下文

行内级格式化上下文用来规定行内级盒子的格式化规则。

先来看看如何生成一个 IFC ：

IFC 只有在一个块级元素中**仅包含**内联级别元素时才会生成。

布局规则

1. 内部的盒子会在水平方向，一个接一个地放置。
2. 这些盒子垂直方向的起点从包含块盒子的顶部开始。
3. 摆放这些盒子的时候，它们在水平方向上的 padding、border、margin 所占用的空间都会被考虑在内。
4. 在垂直方向上，这些框可能会以不同形式来对齐（vertical-align）：它们可能会使用底部或顶部对齐，也可能通过其内部的文本基线（baseline）对齐。
5. 能把在一行上的框都完全包含进去的一个矩形区域，被称为该行的行框（line box）。行框的宽度是由包含块（containing box）和存在的浮动来决定。
6. IFC中的 line box 一般左右边都贴紧其包含块，但是会因为float元素的存在发生变化。float 元素会位于 IFC与与 line box 之间，使得 line box 宽度缩短。
7. IFC 中的 line box 高度由 CSS 行高计算规则来确定，同个 IFC 下的多个 line box 高度可能会不同（比如一行包含了较高的图片，而另一行只有文本）
8. 当 inline-level boxes 的总宽度少于包含它们的 line box 时，其水平渲染规则由 `text-align` 属性来确定，如果取值为 `justify` ，那么浏览器会对 inline-boxes（注意不是inline-table 和 inline-block boxes）中的文字和空格做出拉伸。
9. 当一个 inline box 超过 line box 的宽度时，它会被分割成多个boxes，这些 boxes 被分布在多个 line box 里。如果一个 inline box 不能被分割（比如只包含单个字符，或 word-breaking 机制被禁用，或该行内框受 white-space 属性值为 nowrap 或 pre 的影响），那么这个 inline box 将溢出这个 line box。

那么，IFC 的具体实用在何处呢？

- 水平居中：当一个块要在环境中水平居中时，设置其为 inline-block 则会在外层产生 IFC，通过设置父容器 `text-align:center` 则可以使其水平居中。

> 值得注意的是，设置一个块为 `inline-block` ，以单个封闭块来参与外部的 IFC，而内部则生成了一个 BFC。

- 垂直居中：创建一个IFC，用其中一个元素撑开父元素的高度，然后设置其 `vertical-align:middle` ，其他行内元素则可以在此父元素下垂直居中。

使用 IFC 可以实现多行文本的水平垂直居中，可以看看下面这个例子：

Demo戳我：使用 IFC 可以实现多行文本的水平垂直居中

BFC（Block Formatting Contexts）块级格式化上下文

块格式化上下文（block formatting context）是页面上的一个独立的渲染区域，容器里面的子元素不会在布局上影响到外面的元素。它是决定块盒子的布局及浮动元素相互影响的一个因素。

下列情况将创建一个块格式化上下文：

1. 根元素或其它包含它的元素
2. 浮动 (元素的 float 不为 none)
3. 绝对定位元素 (元素的 position 为 absolute 或 fixed)
4. 行内块 inline-blocks (元素的 display: inline-block)
5. 表格单元格 (元素的 display: table-cell，HTML表格单元格默认属性)
6. 表格标题 (元素的 display: table-caption, HTML表格标题默认属性)

7. overflow 的值不为 visible的元素
8. 弹性盒子 flex boxes (元素的 display: flex 或 inline-flex)

块格式化上下文包括了创建该上下文的元素的所有子元素，但不包括创建了新的块格式化上下文的子元素。

包含浮动元素的块塌缩，清除浮动等都是 BFC 的应用场景。

GFC（Grid Formatting Contexts）栅格格式化上下文

display:grid 篇幅巨大，建议看完大漠老师的教程：

CSS Grid布局

FFC（Flex Formatting Contexts）Flex格式化上下文

如上所述，当 display 的值为 flex 或 inline-flex 时，将生成弹性容器（Flex Containers）。

一个弹性容器为其内容建立了一个新的弹性格式化上下文环境（FFC）。

值得注意的是，弹性容器不是块容器，下列适用于块布局的属性不适用于弹性布局：

1. 在CSS3多列布局模块中定义的 column-* 属性不适用于弹性容器。
2. float 和 clear 属性对于弹性项没有作用，并不会把它带离文档流（或相反）。然而，浮动属性仍然会通过影响display属性的计算值而影响box的生成。
3. vertical-align 属性对于弹性项没有作用
4. ::first-line 和 ::first-letter 伪元素不适用于弹性容器，而且弹性容器不为他们的祖先提供第一个格式化的行或第一个字母。

看看下面的结构，

```
<div class="box">
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</div>
```

```
.box{
  display: flex;
}
```

如上所示，采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。弹性容器中的弹性项（flex item）表示其文档流（in-flow）内容中的框。

`display:flex` 篇幅巨大，建议看完下面两篇：

- 阮一峰--Flex 布局教程
- 弹性块布局

> 借用一句话："'在工作过程中遇到某个属性的使用，浏览器渲染效果与预期效果不符，只能通过死记硬背能避免或应用这种效果，不知晓背后的原理，用**就是这样的**的借口来搪塞自己。"，所以对于布局问题，不要死记硬背，要去理解背后的各种原理。

12、几个特殊且实用的伪类选择器（ :root，:target，:empty，:not ）

每一个 CSS 伪类及伪元素的出现，肯定都是为了解决某些先前难以解决的问题而应运而生的。

学习了解它们，是解决许多其他复杂 CSS 问题或者前沿技术的基础。

这里是 4 个基本的结构性伪类选择器，结构性伪类选择器的共同特征是允许开发者根据文档树中的结构来指定元素的样式。

:root 伪类

`:root` 伪类匹配文档树的根元素。应用到HTML，`:root` 即表示为 `<html>` 元素，除了优先级更高外，相当于html标签选择器。

语法样式

```
:root { 样式属性 }
```

譬如， `:root{background:#000}` ，即可将页面背景色设置为黑色。

由于属于 CSS3 新增的伪类，所以也可以作为一种 HACK 元素，只对 IE9+ 生效。

介绍 `:root` 伪类，是因为在介绍使用 `CSS变量` 的时候，声明全局CSS变量时 `:root` 很有用。

`:empty` 伪类

`:empty` 伪类，代表没有子元素的元素。 这里说的子元素，只计算元素结点及文本（包括空格），注释、运行指令不考虑在内。

考虑一个例子：

```
div{
  height:20px;
  background:#ffcc00;
}
div:empty{
  display:none;
}


<div>1</div>
<div> </div>
<div></div>
```

上述的例子，前两个div会正常显示，而第三个则会 `display:none` 隐藏。

也就是说，要想 `:empty` 生效，标签中连哪怕一个空格都不允许存在。

[Demo戳我：:empty结构性伪类示例]

`:not` 伪类

CSS否定伪类， `:not(X)` ，可以选择除某个元素之外的所有元素。

X不能包含另外一个否定选择器。

关于 `:not` 伪类有几个有趣的现象：

- `:not` 伪类不像其它伪类，它不会增加选择器的优先级。它的优先级即为它参数选择器的优先级。

> 我们知道，选择器是有优先级之分的，通常而言，伪类选择的权重与类选择器（class selectors，例如 `.example` ），属性选择器（attributes selectors，例如 `[type="radio"]` ）的权重相同，但是有一个特例，就是 `:not()` 。 `:not` 否定伪类在优先级计算中不会被看作是伪类，但是在计算选择器数量时还是会把其中的选择器当做普通选择器进行计数。

- 使用 `:not(*)` 将匹配任何非元素的元素，因此这个规则将永远不会被应用。
- 这个选择器只会应用在一个元素上， 你不能用它在排除所有祖先元素。 举例来说， body :not(table) a 将依旧会应用在table内部的 `<a>` 上, 因为 `<tr>` 将会被:not() 这部分选择器匹配。（摘自MDN）

`:target` 伪类

`:target` 伪类，在 `#8、纯CSS的导航栏Tab切换方案` 中已经实践过了，可以回过头看看。

`:target` 代表一个特殊的元素，若是谈论区别的话，它需要一个id去匹配文档URI的片段标识符。

`:target` 选择器的出现，让 CSS 也能够接受到用户的点击事件，并进行反馈。（另一个可以接收点击事件的 CSS 选择器是 `:checked` ）。

13、引人瞩目的 CSS 变量（CSS Variable）

这真是一个令人激动的革新。

CSS 变量，顾名思义，也就是由网页的作者或用户定义的实体，用来指定文档中的特定变量。

更准确的说法，应该称之为 CSS 自定义属性 ，不过下文为了好理解都称之为 CSS 变量。

一直以来我们都知道，CSS 中是没有变量而言的，要使用 CSS 变量，只能借助 SASS 或者 LESS 这类预编译器。

但是新的草案发布之后，直接在 CSS 中定义和使用变量已经不再是幻想了，像下面这样，看个简单的例子：

```
// 声明一个变量：
:root{
  --bgColor:#000;
}
```

> 这里我们借助了上面 #12、结构性伪类 中的 :root{ } 伪类，在全局 :root{ } 伪类中定义了一个 CSS 变量，取名为 --bgColor 。

定义完了之后则是使用，假设我要设置一个 div 的背景色为黑色：

```
.main{
  background:var(--bgColor);
}
```

这里，我们在需要使用之前定义变量的地方，通过 var(定义的变量名) 来调用。

Demo戳我 -- CSS 变量简单示例。

当然，示例正常显示的前提是浏览器已经支持了 CSS 变量，可以看看 CANIUSE。

CSS 变量的层叠与作用域

CSS 变量是支持继承的，不过这里说成级联或者层叠应该更贴切。

> 在 CSS 中，一个元素的实际属性是由其自身属性以及其祖先元素的属性层叠得到的，CSS 变量也支持层叠的特性，当一个属性没有在当前元素定义，则会转而使用其祖先元素的属性。在当前元素定义的属性，将会覆盖祖先元素的同名属性。

其实也就是作用域，通俗一点就是局部变量会在作用范围内覆盖全局变量。

```
:root{
  --mainColor:red;
}

div{
  --mainColor:blue;
  color:var(--mainColor);
}
```

上面示例中最终生效的变量是 --mainColor:blue 。

> 另外值得注意的是 CSS 变量并不支持 !important 声明。

CSS 变量的组合

CSS 变量也可以进行组合使用。看看下面的例子：

```
<div></div>
```

```
:root{
  --word:"this";
  --word-second:"is";
  --word-third:"CSS Variable";
}

div::before{
  content:var(--word)' 'var(--word-second)' 'var(--word-third);
}
```

上面 div 的内容将会显示为this is CSS Variable。

Demo戳我 -- CSS变量的组合使用

CSS 变量与计算属性 calc( )

更有趣的是，CSS 变量可以结合 CSS3 新增的函数 calc( ) 一起使用，考虑下面这个例子：

```
<div> CSS Varialbe </div>
```

```
:root{
  --margin: 10px;
}

div{
  text-indent: calc(var(--margin)*10)
}
```

上面的例子，CSS 变量配合 calc 函数，得到的最终结果是 `text-indent:100px` 。

Demo戳我 -- CSS 变量与 Calc 函数的组合

CSS 变量的用途

CSS 变量的出现，到底解决了我们哪些实际生产中的问题？列举一些：

1、代码更加符合 DRY（Don't repeat yourself）原则。

一个页面的配色，通常有几种主要颜色，同一个颜色值在多个地方用到。之前的 LESS、SASS预处理器的变量系统就是完成这个的，现在 CSS 变量也能轻松做到。

```
:root{
  --mainColor:#fc0;
}
// 多个需要使用到的 --mainColor 的地方
.div1{
  color:var(--mainColor);
}
.div2{
  color:var(--mainColor);
}
```

2、精简代码，减少冗余，响应式媒体查询的好帮手

一般而言，使用媒体查询的时候，我们需要将要响应式改变的属性全部重新罗列一遍。

```
.main {
        width: 1000px;
        margin-left: 100px;
}
@media screen and (min-width:1480px) {
        .main {
                width: 800px;
                margin-left: 50px;
        }
}
```

即便是 LESS 和 SASS 也无法做到更加简便，不过 CSS 变量的出现让媒体查询更加的简单：

```
:root {
  --mainWidth:1000px;
  --leftMargin:100px;
}

.main {
  width: var(--mainWidth);
  margin-left: var(--leftMargin);
}

@media screen and (min-width:1480px) {
        :root {
          --mainWidth:800px;
          --leftMargin:50px;
        }
}
```

看上好像是代码多了，多了一层定义的环节，只是我这里示例的 CSS 改变的样式属性较少，当媒体查询的数量达到一定程度，使用 CSS 变量从代码量及美观程度而言都是更好的选择。

3、方便的从 JS 中读/写，统一修改

CSS 变量也是可以和 JS 互相交互。

```
:root{
  --testMargin:75px;
```

```
}


//  读取
var root = getComputedStyle(document.documentElement);
var cssVariable = root.getPropertyValue('--testMargin').trim();

console.log(cssVariable); // '75px'

// 写入
document.documentElement.style.setProperty('--testMargin', '100px');
```

与传统 LESS 、SASS 等预处理器变量比较

相较于传统的 LESS 、SASS 等预处理器变量，CSS 变量的优点在于:

1. CSS 变量的动态性，能在页面运行时更改，而传统预处理器变量编译后无法更改
2. CSS 变量能够继承，能够组合使用，具有作用域
3. 配合 Javascript 使用，可以方便的从 JS 中读/写

14、为何要规范 CSS 命名方式，及 BEM。

CSS 的命名方式及规范一直处于很混乱的状态，每个团队内部或多或少都有自己的标准。

为什么要规范 CSS 的命名方式？

有过接手别人项目的经历的话，肯定会有这样的感触。就算只是修改几处样式，面对一大堆不是自己写的 CSS ，内心也是崩溃的。

不规范的或者没有统一的命名方式，让你不敢修改别人的 CSS，**他写的这个样式是否有在其他地方引用？这个样式怎么在 DOM 中使用了，但是在样式表中没法找到，能不能删除？**

而良好的 CSS 命名规范，则能有效的规避大部分这些问题。

譬如我们组内遵循的一套命名规范，其中比较重要的一部分：

- 布局：以 `g` 为命名空间，例如： `g-wrap` 、 `g-header` 、 `g-content`
- 状态：以 `s` 为命名空间，表示动态的、具有交互性质的状态，例如： `s-current` 、 `s-selected`
- 工具：以 `u` 为命名空间，表示不耦合业务逻辑的、可复用的的工具，例如： `u-clearfix` 、 `u-ellipsis`
- 组件：以 `m` 为命名空间，表示可复用、移植的组件模块，例如： `m-slider` 、 `m-dropMenu`
- 钩子：以 `j` 为命名空间，表示特定给 JavaScript 调用的类名，例如： `j-request` 、 `j-open`

我觉得没有说哪个规范是最好的，适合自己团队的，能够提高效率的命名规范就是好的。

这里再介绍一下比较受欢迎的 BEM 命名规范。

BEM的意思就是块（block）、元素（element）、修饰符（modifier），是由 Yandex 团队提出的一种 CSS Class 命名方法。

类似于：

```
.block{}
.block__element{}
.block--modifier{}
```

BEM 这种巧妙的命名方法让你的CSS类对其他开发者来说更加透明而且更有意义。BEM代表块（Block），元素（Element），修饰符（Modifier）。上图很好地解释了什么是块，什么是元素。

- 就一个页面来说，开发者其实知道它是由各类模块「块」构成的。这个 Block 并非 inline-block 里的 block，而是所有东西都划分为一个独立的模块，block 是可以相互嵌套的。
- 而「元素」是块中的一部分，具有某种功能。元素是依赖上下文的。
- 它们只有处于他们应该属于的块的上下文中时才是有意义的。「修饰符」则表示块或元素的一些状态，如 hover、active 和 disabled 等。

BEM 中，一个项目中的块名必须是唯一的，明确指出它所描述的是哪个块。相同块的实例可以有相同的名字。一个块范围内的一种元素的名字也必须是唯一的。一种元素可以重复出现多次。

最终使用 BEM 命名出来的 CLASS 样式文件肯定是很不美观，因为使用了单下划线，双下划线，双连接线。但是项目具有一定规模之后，这种命名方式带来的好处也是显而易见的。

BEM 思想的优劣，可以看看知乎大猫的回答：

如何看待 CSS 中 BEM 的命名方式？

15、reset.css 知多少？

大部分的时候，作为前端，我们在写 CSS 样式之前，都知道需要添加一份 `reset.css` ，但是有深究过 `reset.css` 每一句的人恐怕不多，其实其中也是有很多学问的，知己知彼，真正厘清它，对提高 CSS 大有裨益。

reset.css

先来看看早先 YUI 的一个版本的 `reset.css` ，这是一份历史比较悠久的 RESET 方案:

```
body, div, dl, dt, dd, ul, ol, li, h1, h2, h3, h4, h5, h6, pre, form, fieldset, input, textarea, p,
    margin: 0;
    padding: 0;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
fieldset, img {
    border: 0;
}
address, caption, cite, code, dfn, em, strong, th, var {
    font-style: normal;
    font-weight: normal;
}
ol, ul {
    list-style: none;
}
caption, th {
    text-align: left;
}
h1, h2, h3, h4, h5, h6 {
    font-size: 100%;
    font-weight: normal;
}
q:before, q:after {
    content: '';
}
abbr, acronym {
    border: 0;
}
```

首先，我们要知道 CSS RESET 的目的是什么？**是为了消除不同的浏览器在默认样式上不同表现**，但是到今天，现代浏览器在这方面的差异已经小了很多。

reset.css 存在的问题

看看第一段：

```
body, div, dl, dt, dd, ul, ol, li, h1, h2, h3, h4, h5, h6, pre, form, fieldset, input, textarea, p,
    margin: 0;
    padding: 0;
}
```

这一条样式的目的是在于，清除元素的默认 `margin` 和 `padding` 。

但是这一段代码是充满问题的。

- 诸如 div 、dt、li 、th、td 等标签是没有默认 `padding` 和 `margin` 的；
- 如果我现在问你 fieldset 是什么标签，可能没几个人知道，相似的还有如 blockquote 、acronym 这种很生僻的标签，在 html 代码中基本不会出现的，其实没太大必要 RESET ，只会给每个项目徒增冗余代码；

上面的意思是，这一段代码其实做了很多无用功！

要知道，CSS RESET 的作用域是全局的。我们都知道在脚本代码中应该尽量避免滥用全局变量，但是在 CSS 上却总是会忘记这一点，大量的全局变量会导致项目大了之后维护起来非常的棘手。

再看看这一段：

```
h1, h2, h3, h4, h5, h6 {
    font-size: 100%;
    font-weight: normal;
}
ol, ul {
    list-style: none;
}
```

这一段代码，目的是统一了 h1~h6 的表现，取消了标题的粗体展示，取消了列表元素的项目点。

好像没什么问题，但是诸如 h1~h6、ol、ul 这些拥有具体语义化的元素，一旦去掉了它们本身的特性，而又没有赋予它们本身语义化该有的样式（经常没有），导致越来越多人弄不清它们的语义，侧面来说，这也是现在越来越多的页面上 div 满天飞，缺乏语义化标签的一个重要原因。

YUI 版本的 reset 不管高矮胖瘦，一刀切的方式，看似将所有元素统一在同一起跑线上，实则是多了很多冗余代码，得不偿失。

所以，YUI 的 reset.css 的诸多问题，催生出了另一个版本的 reset.css ，名为 **Normalize.css**。

Normalize.css

Normalize.css 有着详尽的注释，由于篇幅太长，可以点开网址看看，本文不贴出全部代码。

> normalize.css v5.0.0

Normalize.css 与 reset.css 的风格恰好相反，没有不管三七二一的一刀切，而是注重通用的方案，重置掉该重置的样式（例如body的默认margin），保留该保留的 user agent 样式，同时进行一些 bug 的修复，这点是 reset 所缺乏的。

Normalize.css 做了什么

Normalize.css 注释完整，每一段代码都说明了作用，总结来说，它做了以下几个工作（摘自官网）：

1. Preserves useful defaults, unlike many CSS resets.
2. Normalizes styles for a wide range of elements.
3. Corrects bugs and common browser inconsistencies.
4. Improves usability with subtle modifications.
5. Explains what code does using detailed comments.

简单翻译一下，大概是：

1. 统一了一些元素在所有浏览器下的表现，保护有用的浏览器默认样式而不是完全清零它们，让它们在各个浏览器下表现一致；
2. 为大部分元素提供一般化的表现；
3. 修复了一些浏览器的 Bug ，并且让它们在所有浏览器下保持一致性；
4. 通过一些巧妙的细节提升了 CSS 的可用性；
5. 提供了详尽的文档让开发者知道，不同元素在不同浏览器下的渲染规则；

真心建议各位抽时间读一读 Normalize.css 的源码，加上注释一共就 460 行，多了解了解各个浏览器历史遗留的一些坑。

关于取舍

那么，最后再讨论下取舍问题。是否 Normalize.css 就真的比 reset.css 好呢？

也不见得，Normalize.css 中重置修复的很多 bug ，其实在我们的项目中十个项目不见得有一个会用得上，那么这些重置或者修复，某种意义上而言也是所谓的冗余代码。

我觉得最重要的是，拒绝拿来主义，不要人云亦云，看见别人说这个 reset.css 好用，也不了解一下，拿来就上到项目中。又或者说写代码几年了，知道每次都引用一个 reset ，却从没有去细致了解其中每一句的含义。

关于维护

当团队根据项目需要（可能混合部分了 reset 或者 normalize ）编写了一份适合团队项目的 reset 之后，随着不断的迭代或者说是复用，很有可能这个版本的 reset.css 会逐渐添加许多其他的全局性的样式，从而又重新陷入上面说的那些问题。

所以我觉得，reset.css 也是需要维护的，关于最佳的 reset.css ，没有一劳永逸的方案，根据项目灵活配置，做出取舍微调，适量裁剪和修改后再使用。

> 最后，搞技术的同学还是应该要有所追求，不要满足于消费别人的总结，一定要去源头看看。