

CSS现状和如何学习

直播的感觉还是有所不同

右上角的偶

接下来要和大家聊的话题

揭开CSS神秘的面纱

CSS是什么？

CSS简单，但不等于CSS容易

CSS的层叠

权重

继承

视觉格式化模型

盒模型

布局

写CSS的姿势

CSS Is Awesome

如何学习CSS

AI会取替CSSer？

在20年岁末有幸和一群优秀大学生在一起聊CSS相关的话题，感到非常的荣幸！在此非常感谢平台（阿里巴巴集团·前端实习生计划）给我这样的机会，同时也要非常感谢好友 裕波，墨苒对我的鼓励和支持。在结束直播的时候，说过后面要整理一篇文字版本，由于个人新转团队（互动团队转到 F(x) Team团队），加上自己太懒，所以今天才开始整理文字版本。所幸能在岁末完成，还算是一种补救。希望接下来的内容，对于新接触CSS或对CSS感兴趣的同学有所帮助，更希望有更多的同学能参与阿里巴巴集团实习生计划的学习，并有所获。

直播的感觉还是有所不同

虽然以前也参加过一些前端行业的会议分享，但在线直播还是平生第一次，感觉还是很不同。其一没有这方面的经验，其二我还是更喜欢线下会议的感觉，因为这样能和大家在一起，有目光的交流。这是一种支持，一种鼓励，一种锻炼，也是一种幸福。

直播之后，我把视频下载下来了，自己也重新看了一回，让我自己感觉有点水，有点啰嗦。此时我更需要感谢那些在听我直播的优秀大学生，真不容易。

在此，奉上直播的视频：

[css-and-damo.mp4](#)

请忽略右上角秃头的偶！

右上角的偶



在前端社区中，大家常称我“大漠”。我个人比较喜欢CSS，在这个领域深耕了近十年，虽无大的成就，但有所小收获。个人平时比较宅，[爱写写博客](#)，看看电影，听听音乐等。

在2014年出版了一本有关于[CSS方面的书](#)：



好像现在没有加印了，网上好象还没有停售，但内容有点偏老，不过还没过时！

在手淘互动呆了五年，最近刚到[淘系前端技术F\(x\) Team团队](#)：

F(x) Team，F(x) 指函数 $F(x)$ ，是机器学习中常出现的符号，深度学习的本质也是求 $f(x)$ 的最优解；意味拥有不同特征的成员经过 fx 团队神奇作用，不断“训练”，一起找到前端智能化团队的最优解。

接下来要和大家聊的话题

01 揭开CSS神秘面纱

02 CSS是什么?

03 CSS简单 $\neq$ CSS容易

04 CSS Is Awesome

05 如何学习CSS?

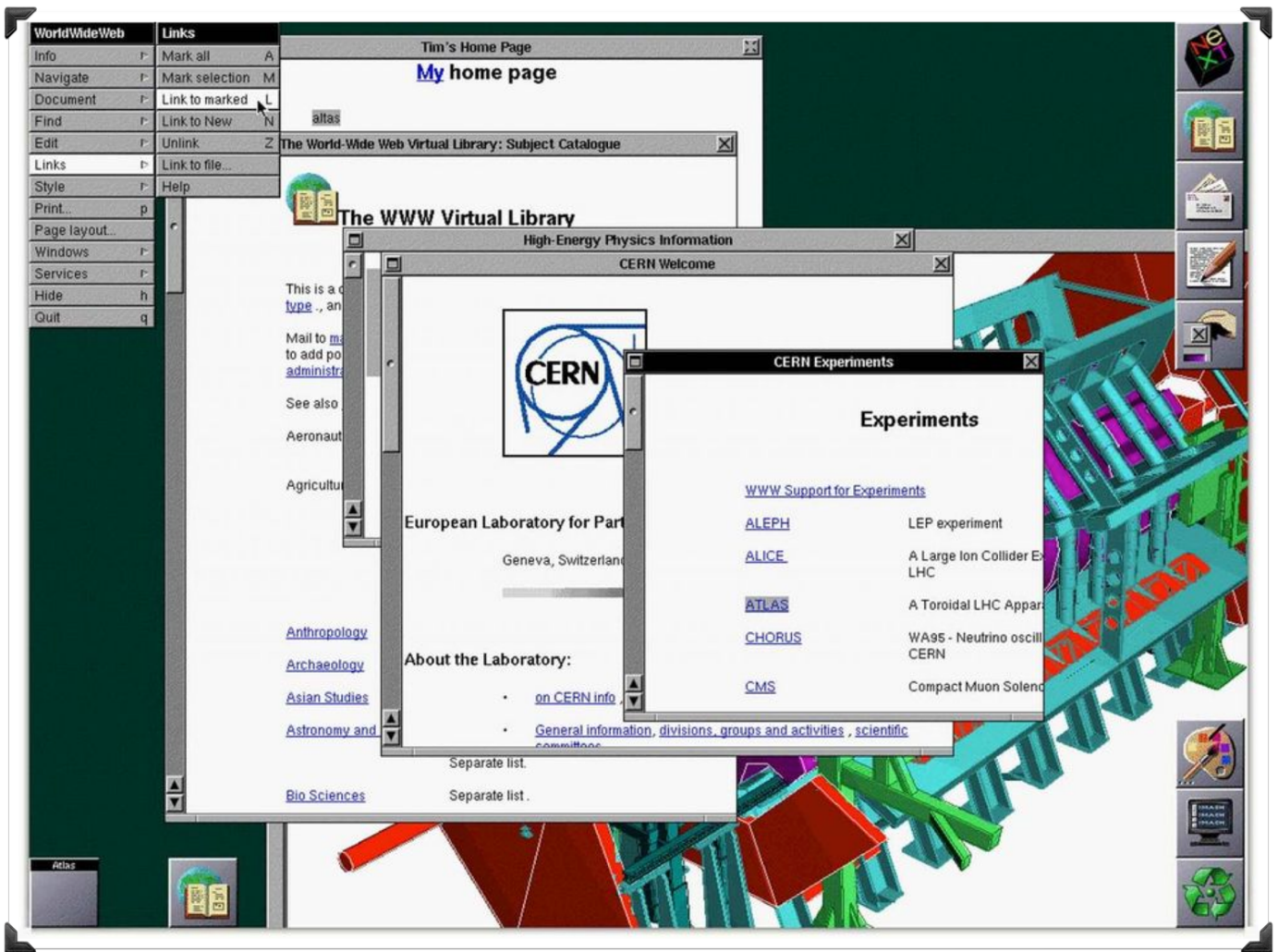
06 阿里前端如何做UI还原?



接下来和大家围绕着这六个方面和大家聊今天的话题。

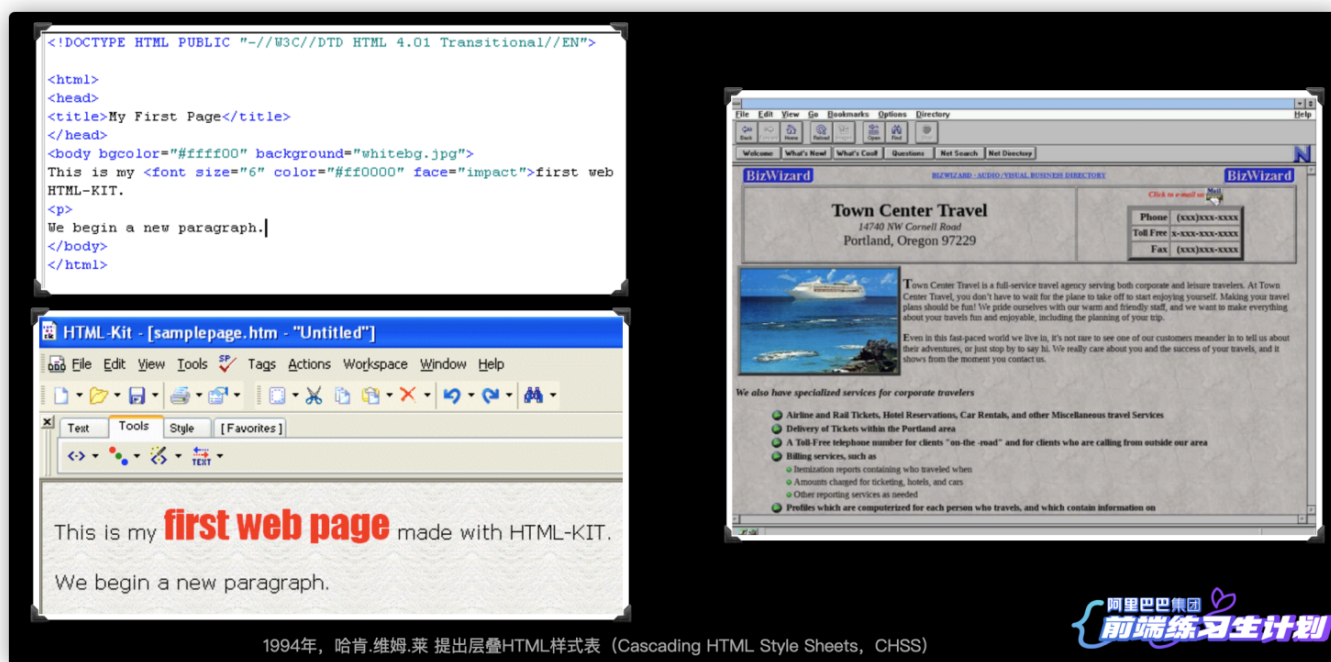
揭开CSS神秘的面纱

自从1994年万维网之父蒂姆·伯纳斯-李在美国麻省理工学院创立了万维网（W3C），至今天也有几十年了。你可能会感到好奇，当初的第一个网页长得像什么：



和今天你所看到的Web页面是不是相差十万八千里。有这种看法是正确的，在那个时候，对于Web页面而言，并没有什么样式一说，而且那个时候Web页面所承担的也就是信息的传递。再简单地说，他只有文字和图像两种，并没有其他的信息元素。

人总是爱美的，即使是在那个年代，人们也想让自己构建的Web页面看上去更美。就在同年（1994年），哈肯·维姆·莱 提出层叠HTML样式表（Cascading HTML Style Sheets, CHSS）：



1994年，哈肯·维姆·莱 提出层叠HTML样式表（Cascading HTML Style Sheets, CHSS）

在HTML的标签上新增了一些属性，比如 `bgcolor` 、 `face` 等，也有一些类似 `<font>` 标签来设置文本样式。

时至今日估计很少有人接触过，或者这样写过Web页面。就我个人而言，多虽没有使用这些东西来构建Web页面，但还是看到过很多页面长成这样。那时候我并不知道，原来这种技术也有自己的专业术语，即 **CHSS（层叠HTML样式表）**。虽说这些标签和属性能让页面上的元素有一些个性化的UI效果，比如改变文本颜色，字体，背景色之类的，但这样的写法对于Web开发者来说是痛苦的，特别对于一个复杂，大型的Web应用，这样的方式来美化Web页面，绝对会让你痛苦难堪。

不说别的，最起码这种方式增加了代码的维护难度，同时也让代码变得臃肿。更没有什么灵活性，可扩展性，可用性一说了。

我们这一批Web开发者是幸运的。就在1994年，哈肯·维姆·莱 和 伯特·波斯合作设计CSS，并在芝加哥的一次会议上展示了CSS的建议。这也是CSS第一次出现在Web开发者（不过，那个时候并没有这个职称）眼中。而且在两年之后，**CSS 1.0的规范出来了**：

Cascading Style Sheets, level 1

W3C Recommendation 17 Dec 1996, revised 11 Apr 2008
superseded on 13 September 2018

This version: <http://www.w3.org/TR/2018/SPSD-CSS1-20180913>
Latest version of CSS: <http://www.w3.org/TR/CSS>
Previous version: <http://www.w3.org/TR/2008/REC-CSS1-20080411>
Authors: [Håkon Wium Lie](mailto:howcome@w3.org) (howcome@w3.org)
[Bert Bos](mailto:bert@w3.org) (bert@w3.org)

Status of this document

This specification is a [Superseded Recommendation](#). A newer specification exists that is recommended for new adoption in place of this specification. New implementations should follow the [latest version](#) of the CSS snapshot.

A list of current W3C Recommendations and other technical documents can be found at <https://www.w3.org/TR/>.

Abstract

This document specifies level 1 of the Cascading Style Sheet mechanism (CSS1). CSS1 is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML documents. The CSS1 language is human readable and writable, and expresses style in common desktop publishing terminology.

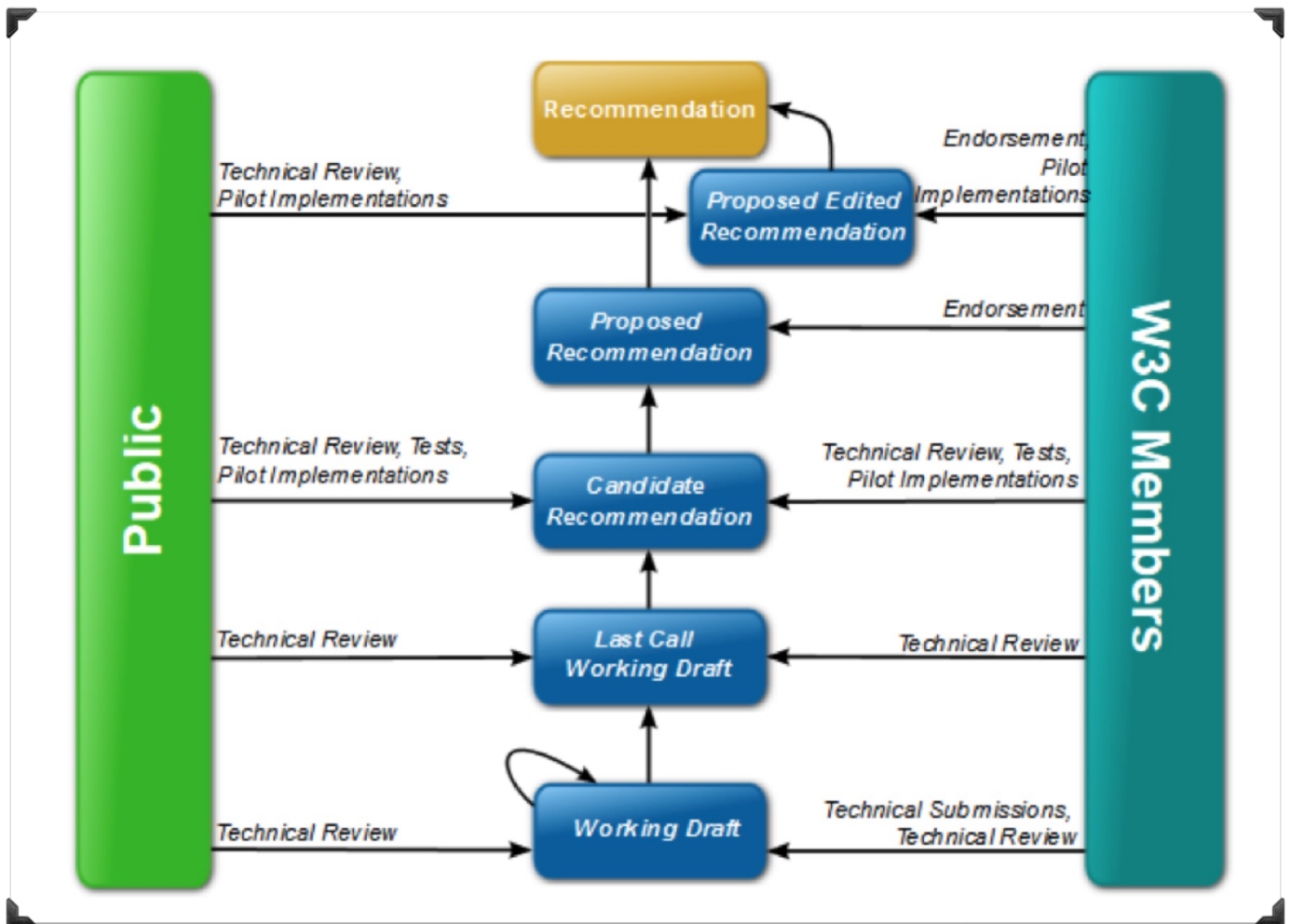
One of the fundamental features of CSS is that style sheets cascade; authors can attach a preferred style sheet, while the reader may have a personal style sheet to adjust for human or technological handicaps. The rules for resolving conflicts between different style sheets are defined in this specification.

而且经历两年的发展，有了CSS2.0版本，在这之后，还经历了CSS 2.1和CSS 2.2版本的迭代。同时CSS 2.1规范指导Web开发者写CSS很多年。直到后面，也就是大约2015年，W3C规划的CSS工作小组发现CSS发展的越来越快，有关于CSS方面的特性增加了不少，而且不同的特性推进速度都有所不同。也就这个时候，W3C的CSS工作小组为了能更好的维护和管理CSS的特性，该组织决定不在以CSS的版本号，比如我们熟悉的CSS1.0、CSS2.0、CSS2.1这样的方式来管理CSS。而是将每个CSS功能特性拆分成独立的功能模块，并且以Level 1, Level2, Level 3等方式来管理CSS规范中的特性：



正如下图所示，即使CSS按功能模块拆分向前推进，但也并不代表拆分之后的CSS特性模块就能很快的进入到正式的标准规范中，就好比CSS2.0到正式进入标准规范也历时很长。这主要是因为，**任何一个CSS的特性到成为真正的规范要经历多个阶段**：

- **编辑草案（ED）**：初始阶段，非常粗糙
- **首个公开工作草案（FPWD）**：首个公开版本，开始接受工作组反馈
- **工作草案（WD）**：会吸收来自工作组和社区的反馈，浏览器早期实现通常从这个阶段开始
- **候选推荐规范（CR）**：相对稳定版本，比较适合实现和测试
- **提名推荐规范（PR）**：W3C会员公司对这项规范表达反对意见的最后机会
- **正式推荐规范（REC）**：一项W3C技术规范的最终阶段



这个时候你可能已经感觉到了，一个东东到成为正式规范是多么漫长的。这个漫长的过程除了W3C规范组织自己的问题，也还有客户端厂商为了各自的利益，在特性的支持上也有所不同。对于我们这些Web开发者来说也是有问题，因为你们更多的时候只是CSS特性的使用者或者说布道者，但并不是推动者。对于我们身在国内的Web开发者而言，很少有人直接参与CSS特性的制定，讨论和推进。

在CSS 选择器 Level4模块出来的时候，社区开始有CSS4这个说法。事实上呢，早在2016年 @Rachelandrew就在社区中发声过《[Why there is no CSS4 – explaining CSS Levels](#)》：



在Github上@Jen Simmons 在 [CSSWG Drafts](#)上单独开了一个关于CSS4相关的帖子，社区中一些大神都对这个话题发表了自己的观点：

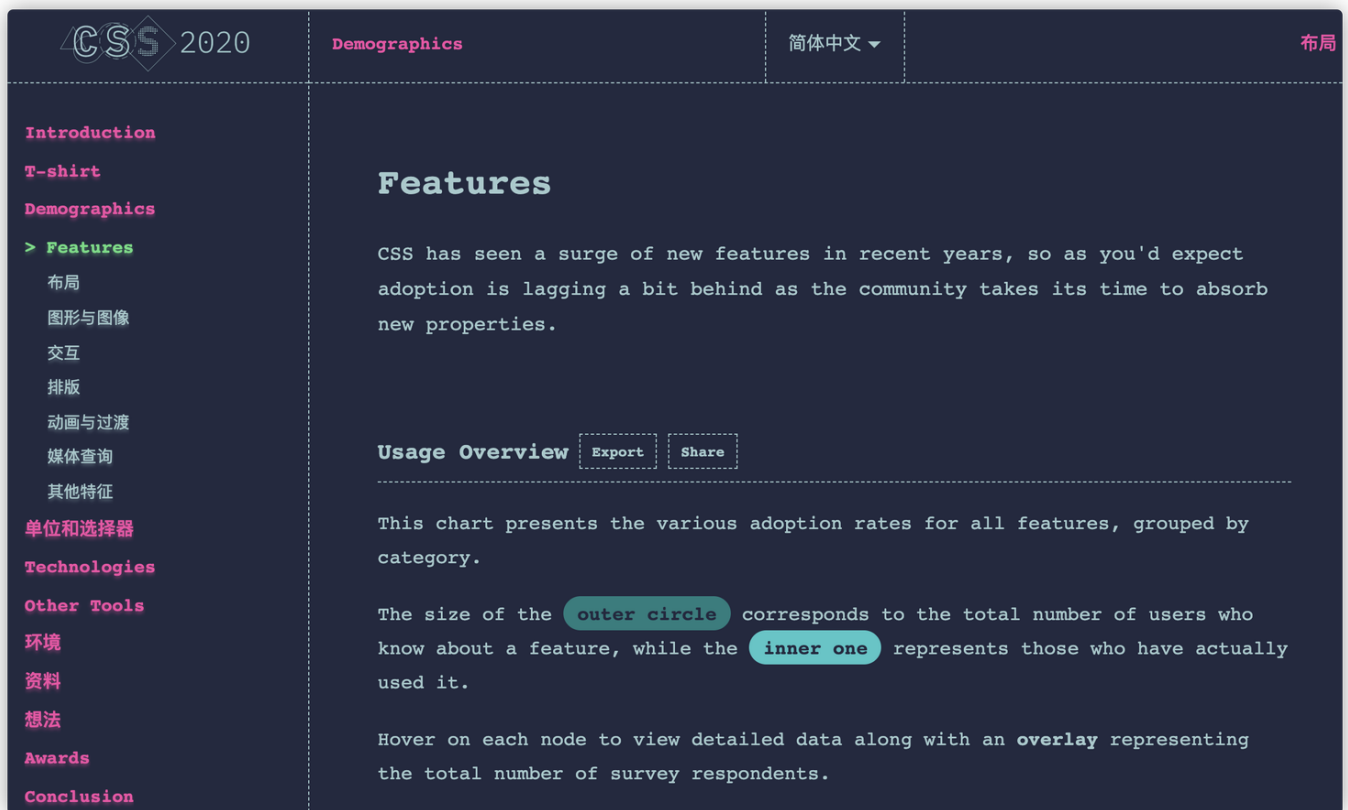
- [Why Are We Talking About CSS4?](#)
- [There Is No Such Thing As CSS4](#)
- [Why There Is No CSS4: Explaining CSS Levels](#)
- [Where Is CSS4? When Is It Coming Out?](#)
- [CSS4: La nueva versión de CSS que nunca existirá](#)

上面主要和大家聊了CSS的历史以及她的发展历程，针对这方面的话题，我在自己的[图解CSS](#)系列中[第一章](#)就是讨论这个的。

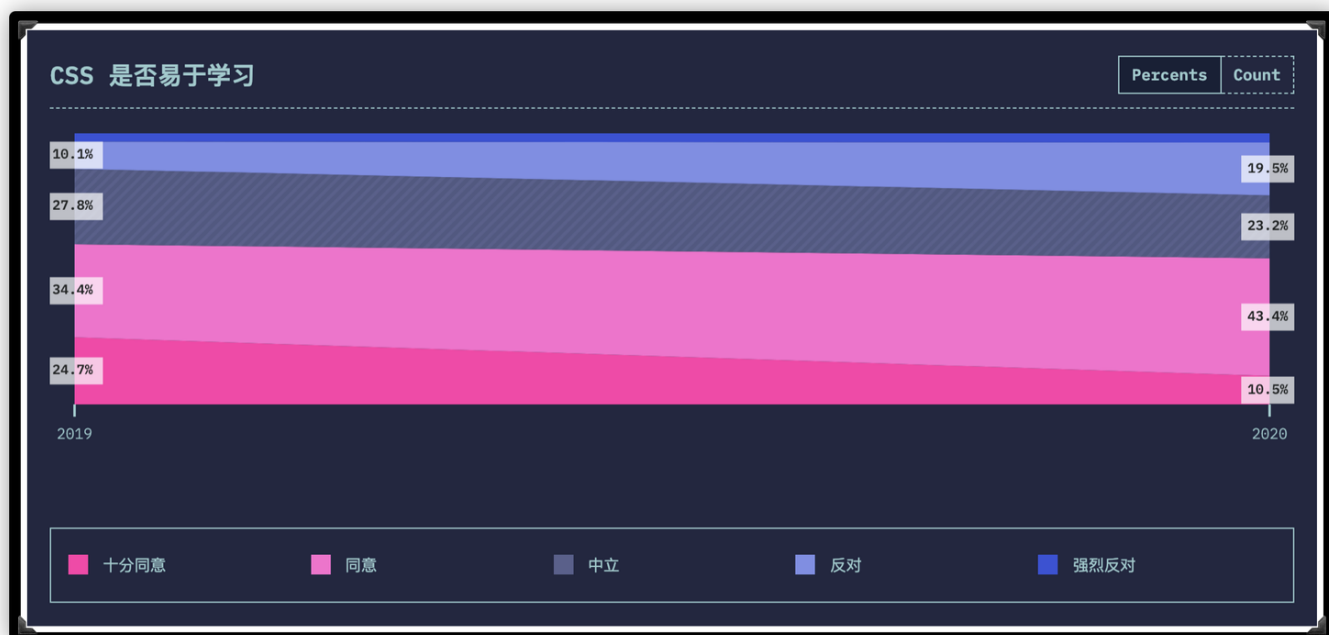
聊完CSS发展之后，我们来再来聊聊CSS的现状。从[2019年](#)和[2020年](#)CSS发展状态报告中，我们就可以看到CSS发展的一个现状：



这份报告相对于来说还是比较详细的：



除了对CSS特性、单位和选择器、方法论、框架、工具、环境等有统计之外，还对使用CSS的人群及现状也有描述，比如：



除了CSS状态发展报告之外，还有[每年的Web年鉴](#)对CSS也会有相应的调查统计：

Index

Introduction

Methodology

Usage

Selectors and the cascade

Class names

IDs

!important

Part I Chapter 1

CSS [UNEDITED]



Written by [Lea Verou](#), [Chris Lilley](#), and [Rachel Andrew](#)

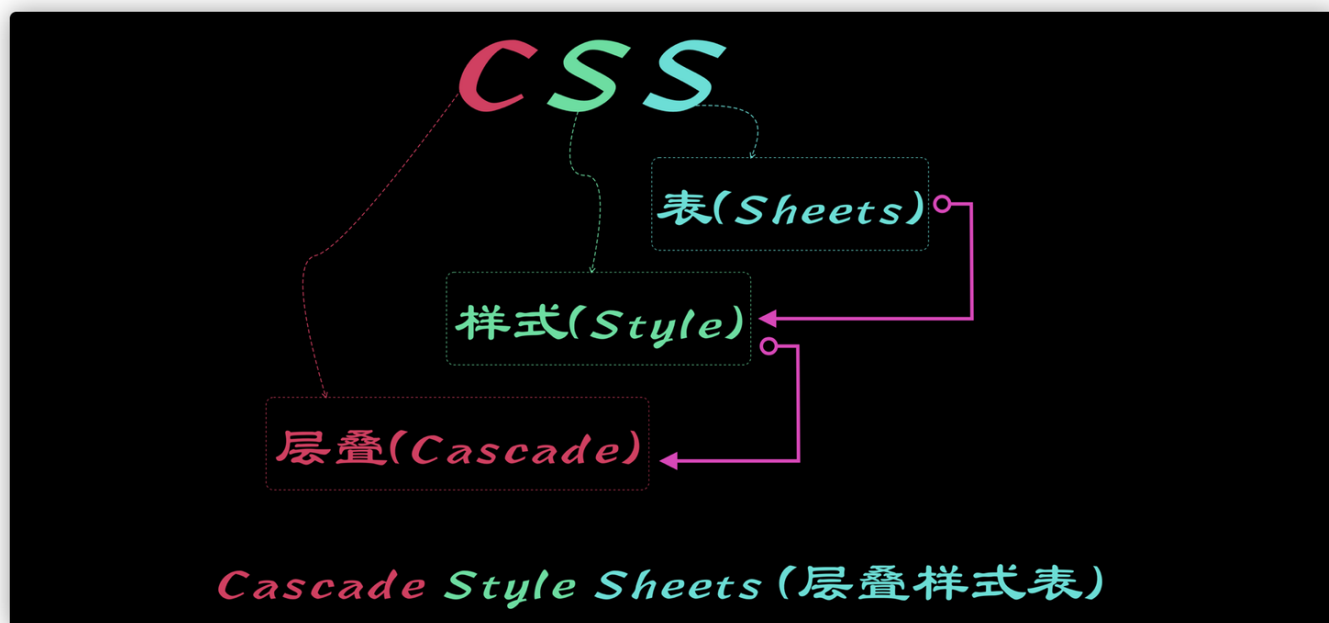
Reviewed by [Estelle Weyl](#), [Elika Etemad aka fantasai](#), [Jens Oliver Meiert](#), [Miriam Suzanne](#), [Catalin Rosu](#), and [Andy Bell](#)

Analyzed by [Rick Viscomi](#), [Lea Verou](#), and [Pokidov N. Dmitry](#)

另外，还有一些其他的调查报告也会有一些CSS相关的报告，但都没有CSS状态发展报告这样详细和针对性。如果你对其他报告中关于CSS方面的调查分析，可以阅读 @[Geoff Graham](#) 的《[2020 Roundup of Web Research](#)》一文中列出的报告中查阅：

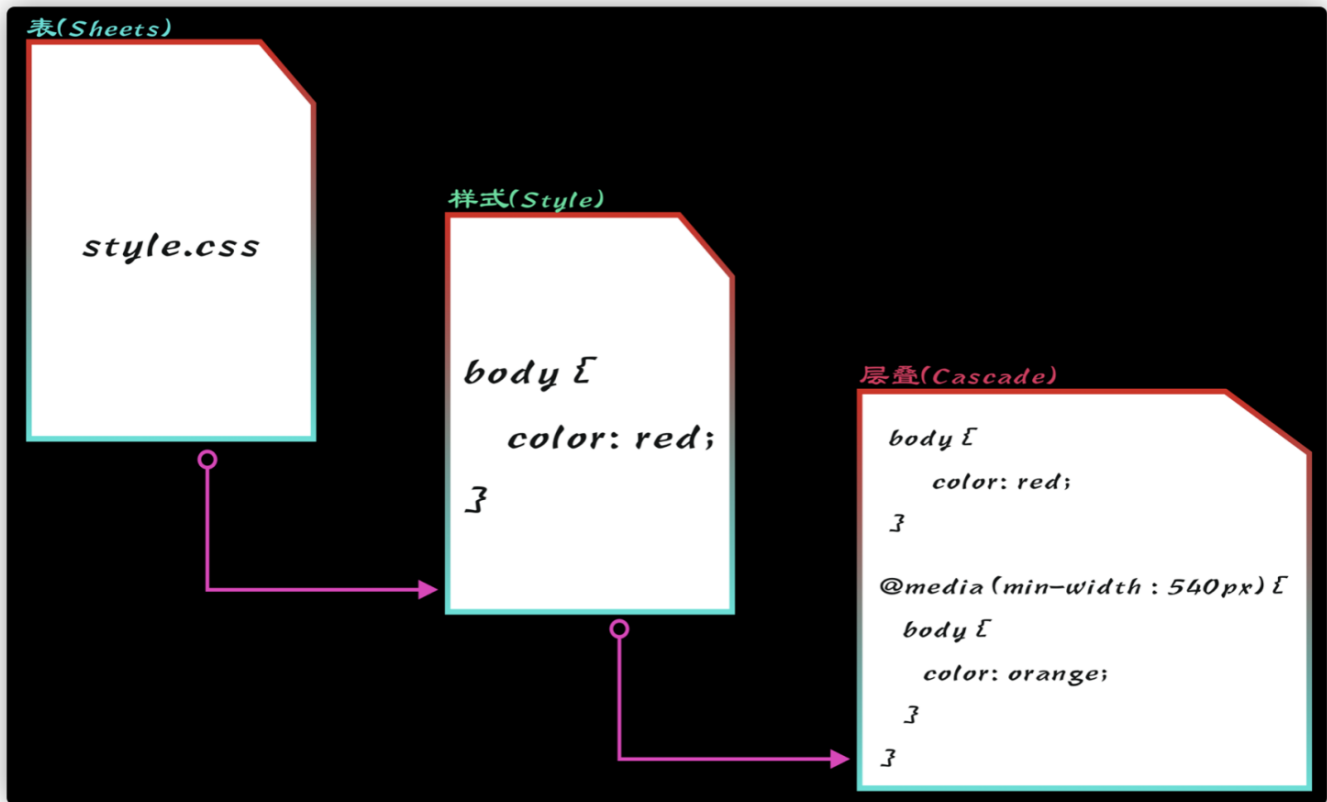
- Part 1:** [2020 MDN Web Developer Needs Assessment](#)
- Part 2:** [HTTP Archive's annual state of the web report](#)
- Part 3:** [State of CSS 2020](#)
- Part 4:** [GitHub's 2020 State of the Octoverse](#)
- Part 5:** [Google's Year in Search](#)
- Part 6:** [Stack Overflow Developer Survey 2020](#)
- Part 7:** [Angular Developer Survey 2020](#)
- Part 8:** [JetBrains State of Developer Ecosystem 2020](#)
- Part 9:** [The WebAIM Million 2020](#)
- Part 10:** [CodinGame 2020 Developer Survey](#)
- Part 11:** [LinkedIn 2020 Workplace Learning Report](#)
- Part 12:** [HackerRank Developer Skills Report](#)
- Part 13:** [Microsoft IoT Signals Report](#)
- Part 14:** [Developer Economics Survey](#)
- Part 15:** [UpWork's Freelance Forward 2020](#)
- Part 16:** [UXTools.co 2020 design tools survey](#)
- Part 17:** [Postman's 2020 State of the API Report](#)
- Part 18:** [Ionic Framework 2020 Developer Study](#)
- Part 19:** [Mailchimp's 2020 Annual Report](#)
- Part 20:** [Campaign Monitor's Email Marketing Benchmarks](#)

CSS是什么？



CSS称为 层叠样式表，即是 Cascade Style Sheets三个词的首字母缩写。每个字母代表的含义不同：

- **C (Cascade)**：指的是层叠，在CSS中编写样式规则是一个一个排列下来，可以简单的理解为先后顺序
- **S (Style)**：第一个S，它指的是样式规则，比如 `body{color: red}`
- **S (Sheets)**：第二个S，它指的是样式表，就是我们常说的 `.css` 文件，CSS的代码会放置在样式表里



Web开发者都知道，一个Web页面或Web应用都由HTML、CSS和JavaScript三个部分组成：

HTML

Structure



CSS

Style



JavaScript

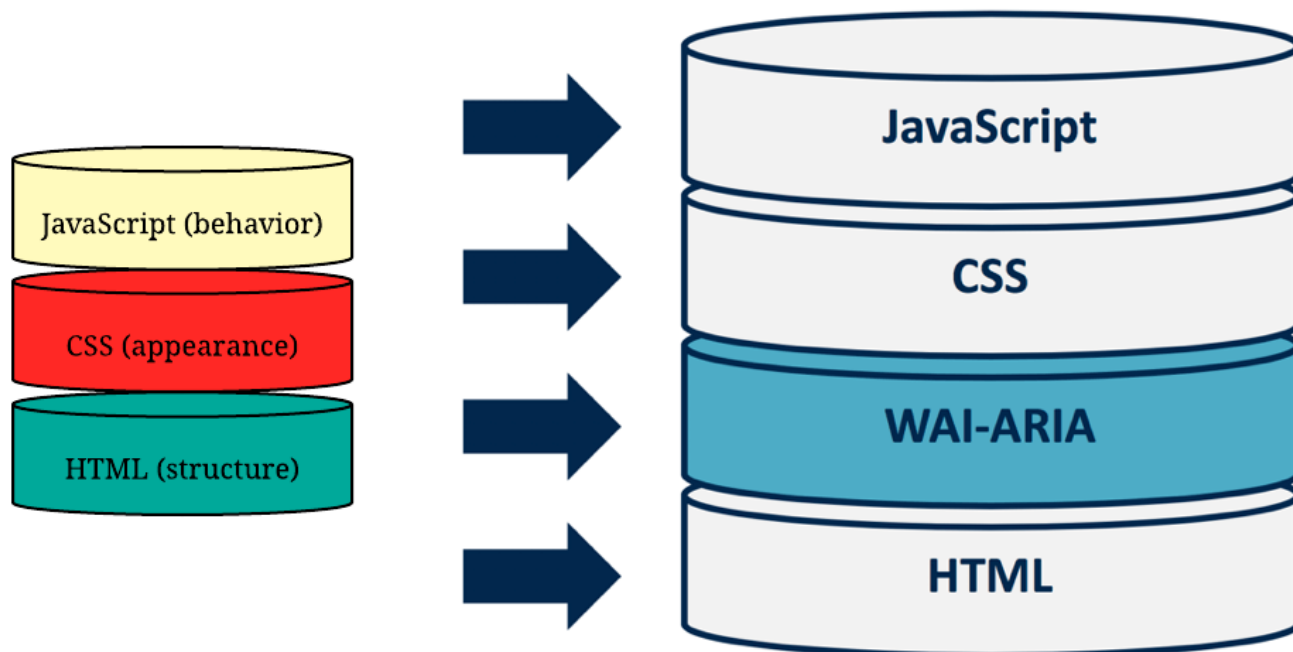
Behaviour



每个部分在Web中所起的作用都有所不一样：

- **HTML**：是Web应用的骨架
- **CSS**：是用来美化Web的
- **JavaScript**：是用来控制Web元素中的交互行为的

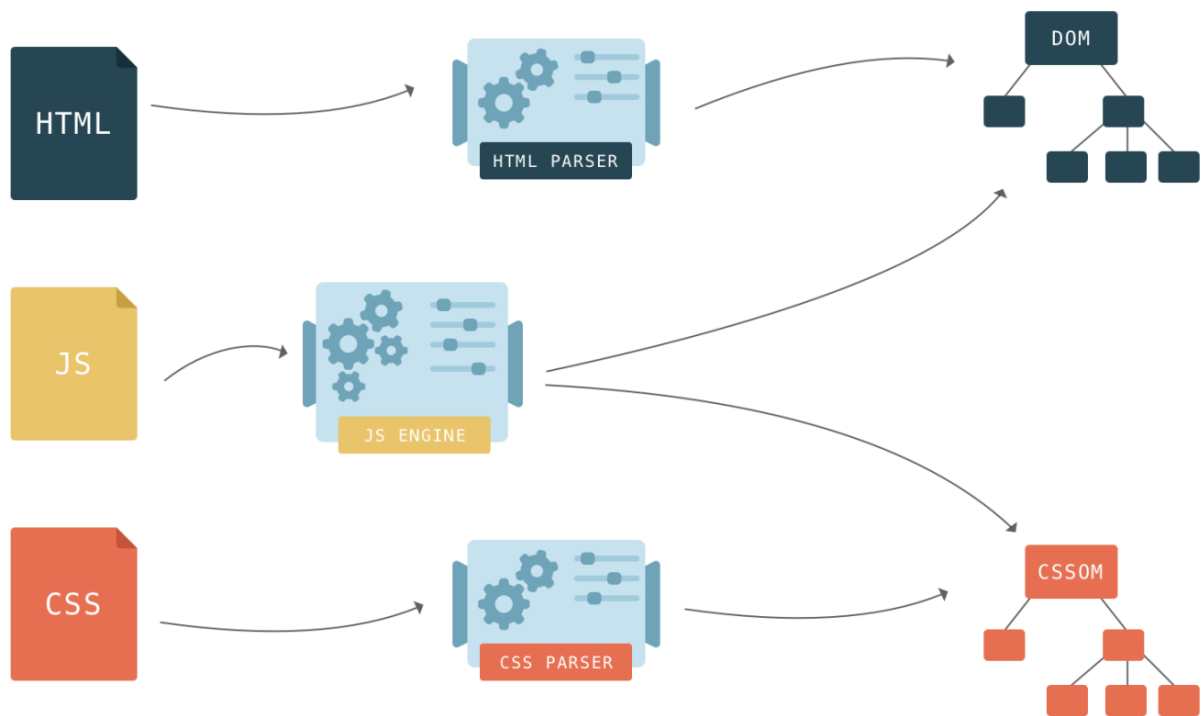
时至今日，Web应用或Web页面除了我们上面所说的HTML、CSS、JavaScript三个部分组成之外，它还有一个新的部分，即 [WAI-ARIA](#)：



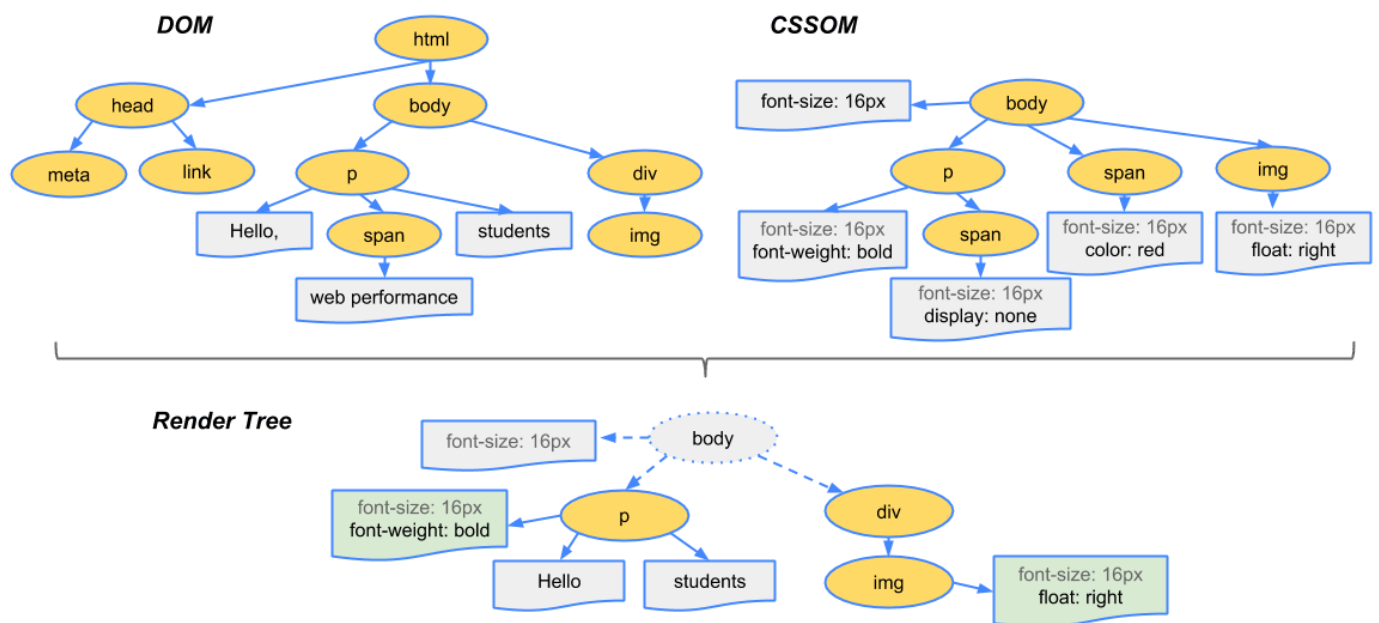
WAI-ARIA 是Web可访问性（A11Y）规范中的一部分，主要用来帮助我们构建一个更具可访问性的Web应用。如果你对这方面知识感兴趣的话，[可以阅读 A11Y系列相关的教程](#)。

对于初次接触CSS的同学来说，肯定会感到好奇，CSS是如何工作的。要详细的讲清楚，那所涉及到的东西就多了，其中就少不了关于 [浏览器工作原理](#) 相关的内容。这方面也不是我擅长的领域，我只能在此抛砖引玉。

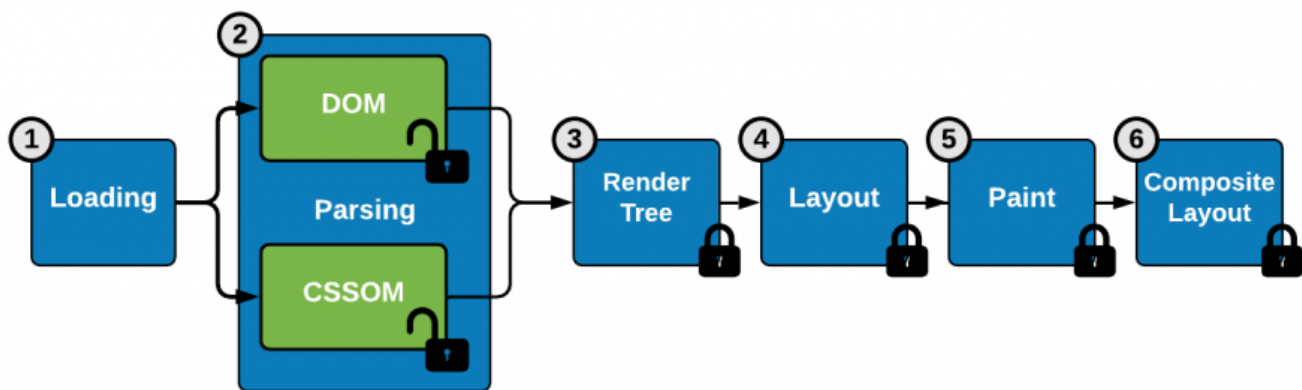
前面提到过，Web页面至少由HTML、CSS和JavaScript三个部分构成，其中HTML会经过HTML Parser将HTML结构转换成**DOM Tree**；CSS会经过CSS Parser将CSS转换成**CSSOM Tree**，正如下图所示：



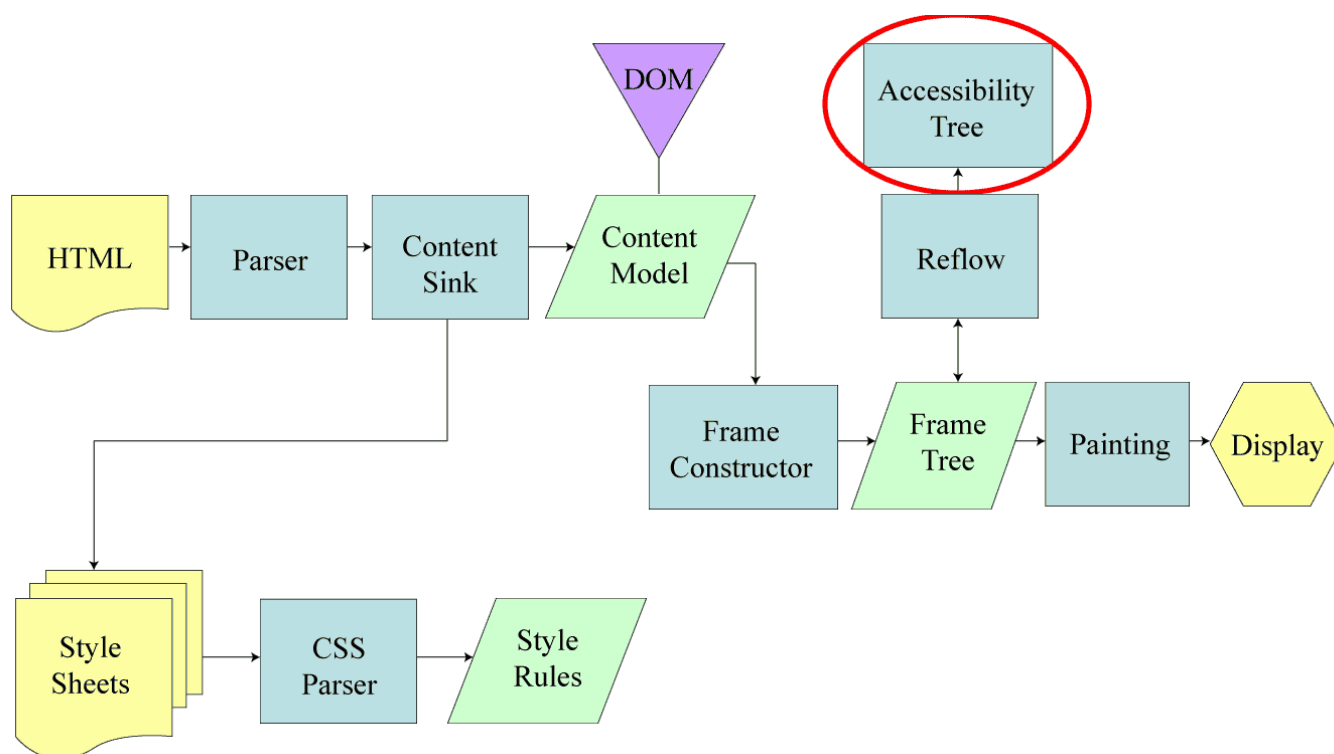
DOM树和CSSOM树将会构建出渲染树：



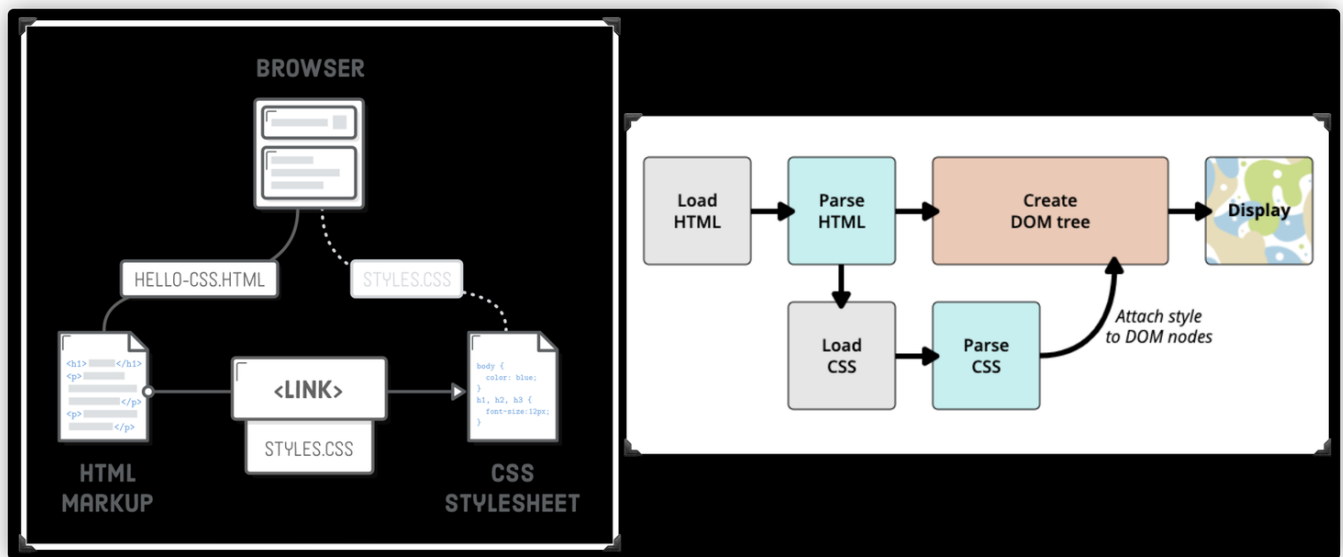
最终经过几个过程就渲染出我们所能看到的Web页面：



如果再把AOM（可访问树）引入起来的话，大致就像下图这样：



这有点偏离我们今天聊的主题了。如果你其中过程不想了解太多的话，只需要简单地记作：“CSS样式要和相应的HTML元素结合在一起使用，才能在浏览器渲染出来，呈现给用户”。



就我个人而言，我更喜欢把CSS称为七巧板或积木：



实现同一效果，我们可以通过多种不同方式来完成。

也正因为如此，很多同学都认为CSS很烦人：



CSS简单，但不等于CSS容易

SIMPLE $\neq$ EASY

@Jeremy Keith 在2017年的CSS Day 大会说过：

Unlike a programming language that requires knowledge of loops, variables, and other concepts, CSS is pretty easy to pick up. Maybe it's because of this that it has gained the reputation of being simple. It is simple in the sense of "not complex", but that doesn't mean it's easy. Mistaking "simple" for "easy" will only lead to heartache. — @Jeremy Keith



大致意思是：

CSS与其编程语言不同，她没有循环、逻辑和其他概念，它只是声明式的语言，因此，CSS很容易上手。也许正是因为如此，它才获得了简单的美誉。在 "不复杂 "的意义上，它是简单的，但这并不意味着它很容易。把 "简单 "误认为是 "容易"，只会让人心痛”

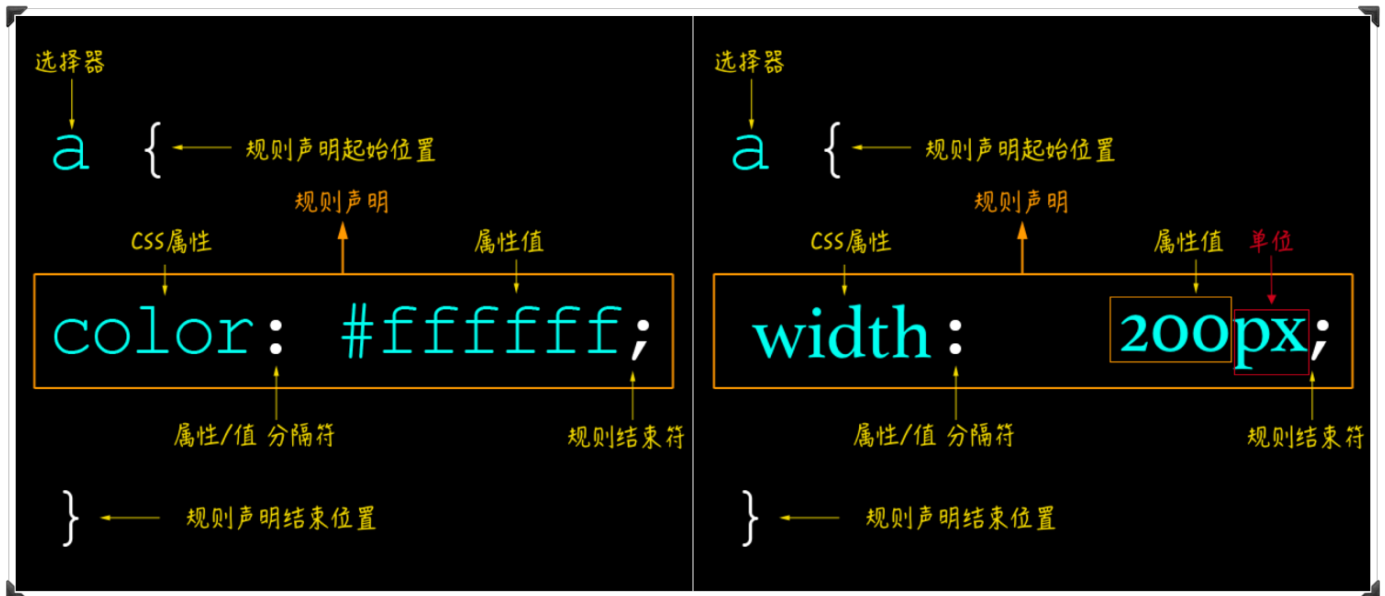
就此话题，在Twitter上，或者在社区时不时的就会有所争论：



有时候在Twitter上你也会感到，原来全世界的前端圈都像是个娱乐圈！

那么为什么会有这些争论呢？为此我讲讲我自己对这方面的看法。

先上一张图：



如果从CSS的语法规则和属性的使用上来说，它的确非常地简单，因为你只需要知道选择器选中了HTML中的哪个元素，给他运用了什么样的属性和值，应该能在浏览器中呈现出来，比如：

```
1 body {  
2     color: #fff;  
3     background-color: #09f;  
4 }
```

上面几行简单的CSS代码，就可以让你在浏览器中看到一个蓝底白字的效果：



CSS is Simple, But not ease!

而且也不需要一个一个把所有CSS属性及属性的值都记在脑中，因为我们有很多方式可以查阅到，比如[MDN上就有CSS属性列表](#)：

-	• font-stretch	• padding-inline-end
• -webkit-line-clamp	• font-stretch (@font-face)	• padding-inline-start
A	• font-style	• padding-left
• :active	• font-style (@font-face)	• padding-right
• additive-symbols (@counter-style)	• font-synthesis	• padding-top
• ::after (:after)	• font-variant	• @page
• align-content	• font-variant (@font-face)	• page-break-after
• align-items	• font-variant-alternates	• page-break-before
• align-self	• font-variant-caps	• page-break-inside
• all	• font-variant-east-asian	• paint()
• <an-plus-b>	• font-variant-ligatures	• paint-order
• <angle>	• font-variant-numeric	• path()
• <angle-percentage>	• font-variant-position	• pc
• animation	• font-variation-settings	• <percentage>
• animation-delay	• font-variation-settings (@font-face)	• perspective
• animation-direction	• font-weight	• perspective()
• animation-duration	• font-weight (@font-face)	• perspective-origin
• animation-fill-mode	• format()	• place-content
• animation-iteration-count	• fr	• place-items
• animation-name	• <frequency>	• place-self
• animation-play-state	• <frequency-percentage>	• ::placeholder
• animation-timing-function	• :fullscreen	• pointer-events
• @annotation	G	• polygon()
• annotation()	• gap	• <position>
• attr()	• grad	• position
	• <gradient>	• prefix (@counter-style)
		• pt
		• px

这就是为什么说CSS简单的主要原因。

那么为什么说CSS不容易呢？这是有很多原因的，先拿大家认为CSS简单的规则来说：

```

1 <div class="card">1</div>
2
3 <div class="card__news">2</div>

```

```

1 .card {
2     width: 100px;
3 }
4

```

```
5 .card.card__news {  
6     width: 100%;  
7 }
```

想象一下，这两个 `div` 的 `width` 分别是什么呢？试问一下，你自己一看到上面的代码能自信，马上说出正确的答案？如果你心理想的答案和自己验证得到的答案是一致的，那么恭喜你，你对CSS有一定的认知，至少不是菜鸟级。那么请继续往下看这个示例：

```
1 <div class="a">Text</div>  
2 <div class="b">Text</div>  
3 <div class="a b">Text</div>  
4 <div class="b a">Text</div>
```

```
1 .a {  
2     color: red;  
3 }  
4  
5 .b {  
6     color: blue;  
7 }  
8  
9 .a.b {  
10     color: orange;  
11 }  
12  
13 .b.a {  
14     color: yellow;  
15 }
```

试问这四个 `div` 的文本颜色分别是什么？印象中 [@Max Stoiber](#) 在twitter上也出过类似的一个问卷：

 推文



Max Stoiber

@mxstbr



How well do you know CSS? 🧑🏫

Given these classes:

```
.red {  
  color: red;  
}
```

```
.blue {  
  color: blue;  
}
```

Which color would these divs be?

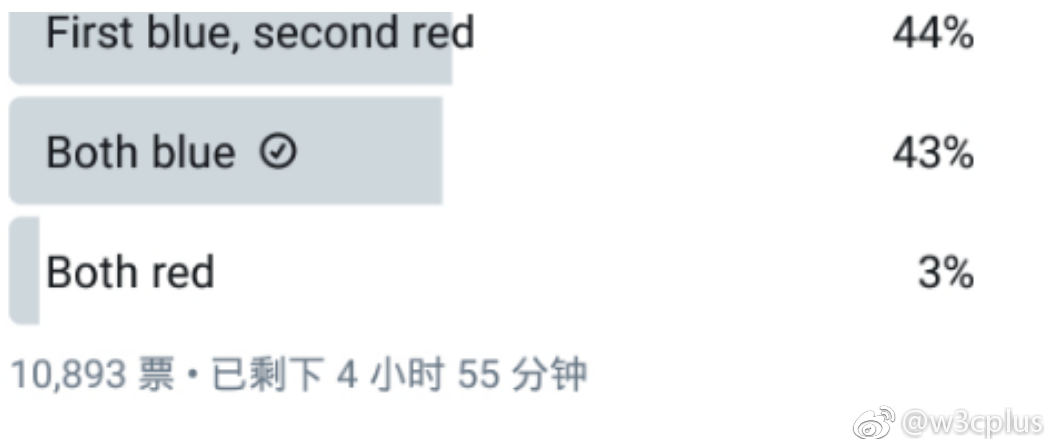
```
<div class="red blue">
```

```
<div class="blue red">
```

[翻译推文](#)

First red, second blue

9%



就此题可以让不少同学感到困惑。除此之外，CSS中还有很多基本概念，可以让你不再觉得CSS容易，比如：

- 层叠
- 继承
- 权重
- 视觉格式化模型
- 盒模型
- 布局
- 样式维护
- 等等

上面列的这些基础概念，每一个都需要一篇或多篇长文才能彻底讲清楚。所以我只能在此为大家抛砖引玉，并在相应的内容上多附上相关知识点的链接。感兴趣的同学，可以点击相应的链接进行扩展阅读。

CSS的层叠

按照前面的解释，层叠是指“**Cascade**”，拿个示例来解说可能会更易于理解。比如我们在一个

`style.css` 样式文件中，对同一个元素写了两个样式规则：

```
1 <a href="#" class="link">Link</a>
```

```
1 a {  
2     color: #90f;  
3 }
```

```
4
5 .link {
6     color: #f36;
7 }
```

a 先于 `.link` 出现，就是对同一元素使用的样式规则出现的先后顺序，被称为层叠。

事实上，CSS中的层叠要比这个更为复杂，除了前面示例中提到的样式规则出现的顺序还有另一种层叠之说，它会涉及到CSS其他的一些概念，比如说：

- 文档流 (Normal Flow)
- 格式化上下文 (Formatting Context)
- 层叠上下文 (Stacking Context)
- 层叠水平 (Stacking Level)
- 层叠顺序 (Stacking Order)

也基于这个原因，我把CSS中的层叠分为两个部分来阐述：

- ① 文档流
- ② 格式上下文
- ③ 层叠上下文
- ④ 层叠顺序
- ⑤ 样式层叠



先来说第一部分的。

文档流

在CSS中，文档流是一个很基础也是很重要的一个概念。很多时候她被称为Document Flow，但在CSS的标准被称为Normal Flow，即普通流或常规流。大家更喜欢称之为文档流。那么CSS的文档流是怎么回事呢？

在HTML中任何一个元素其实就是一个对象，也是一个盒子。在默认情况下它是按照出现的先后顺序来排列，而这个排列的顺序就是文档流。

文档流是元素在Web页面上的一种呈现方式。所有的HTML元素都是块盒子（Block Boxes，块级元素）或行内框（Inline Boxes，行内元素）。当浏览器开始渲染HTML文档时，它从窗口的顶端开始，经过整个文档内容的过程中，分配元素需要的空间。除非文档的尺寸被CSS规则限定，否则浏览器垂直扩展文档来容纳全部的内容。每个新的块级元素渲染为新行。行内元素则按照顺序被水平渲染直到当前行遇到边界，然后换到下一行垂直渲染。

事实上，在普通文档流中的盒子属于一种格式化上下文（Formatting Context），大家较为熟悉的就是块格式化上下文（Block formatting context）和行内格式化上下文（Inline formatting context）。不过有一点面要注意，它们只能是其中一者，但不能同时属于两者。言外之意，任何被渲染的HTML元素都是一个盒子（Box），这些盒子不是块盒子就是行内盒子。即使是未被任何元素包裹的文本，根据不同的情况，也会属于匿名的块盒子或行内盒子。

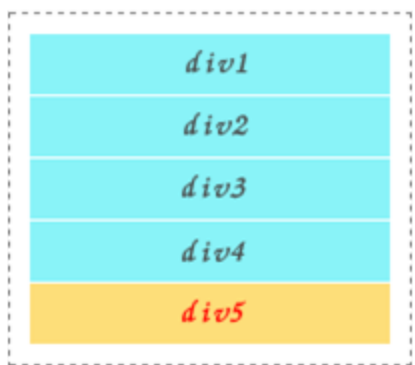
综合上面的描述，也可以理解格式化上下文对元素盒子做了一定的范围的限制，其实就是类似有一个width和height做了限制一样。如果从这方面来理解的话，普通流就是这样的一个过程：

- 在对应的块格式化上下文中，块级元素按照其在HTML源码中出现的顺序，在其容器盒子里从左上角开始，从上到下垂直地依次分配空间层叠（Stack），并且独占一行，边界紧贴父盒子边缘。两相邻元素间的距离由margin属性决定，在同一个块格式化上下文中的垂直边界将被重叠（Collapse margins）。除非创建一个新的块格式化上下文，否则块级元素的宽度不受浮动元素的影响。
- 在对应的行内格式化上下文中，行内元素从容器的顶端开始，一个接一个地水平排列。

扯了这么多，如果简单的描述就是：如何排列HTML元素而已。拿个块格式化上下文的普通文档流来举例，就像下面这样：

```
1 <div>1</div>
2 <div>2</div>
3 <div>3</div>
4 <div>4</div>
5 <div>5</div>
```

对应的效果如下：



格式化上下文

前面多次提到 格式化上下文 ，在CSS中，格式化上下文指的是：

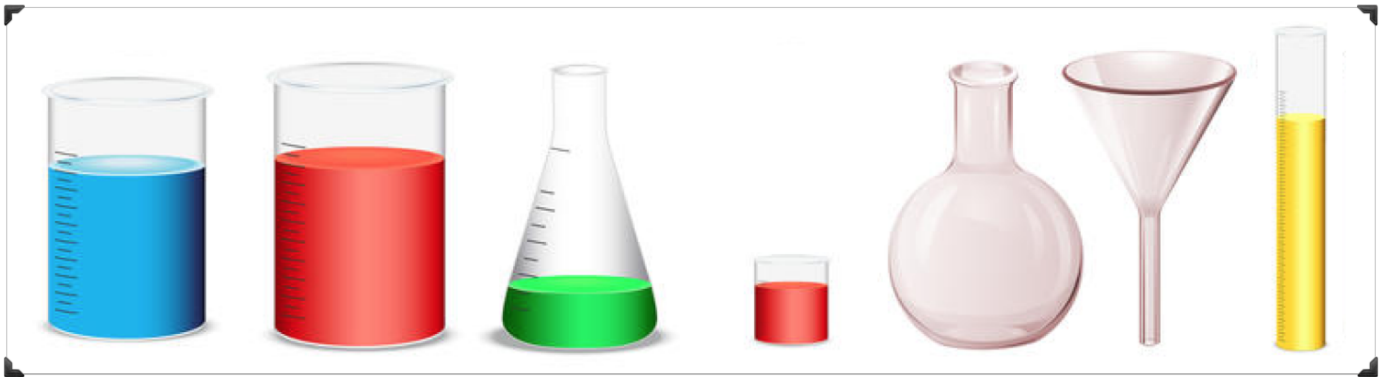
初始元素定义的环境

其主要包含两个要点，一个是 元素定义的环境 ，另一个是 初始化。

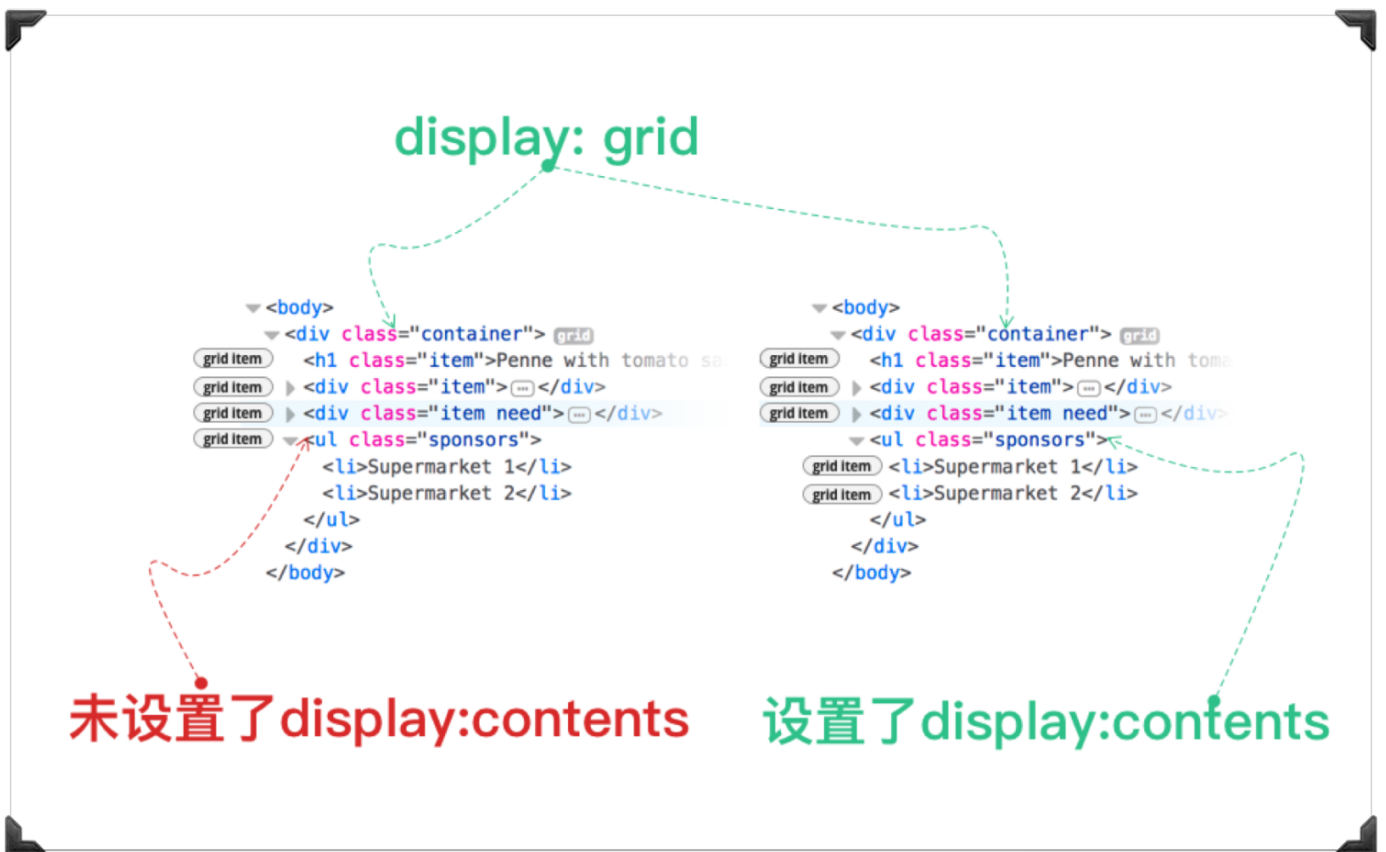
我们使用CSS的 `display` 属性可以对元素进行格式化，即 创建格式化上下文 。

单个值	两个值	描述
block	block flow	正常流内的块级盒子
inline	inline flow	正常流内的内联级盒子
inline-block	inline flow-root	定义一个BFC的内联级盒子
list-item	block flow list-item	正常文档流和带有附加标记的块级盒子
flow-root	block flow-root	定义一个BFC的块级盒子
flex	block flex	带有内部伸缩布局的块级盒子
inline-flex	inline flex	带有内部伸缩布局的内联级盒子
grid	block grid	带有内部网格布局的块级盒子
inline-grid	inline grid	带有内部网格布局的内联级盒子
table	block table	带有内部表格布局的块级盒子
inline-table	inline table	带有内部表格布局的内联级盒子
none		从盒子树中移除，包括其所有后代元素
contents		元素替换为框树中的内容

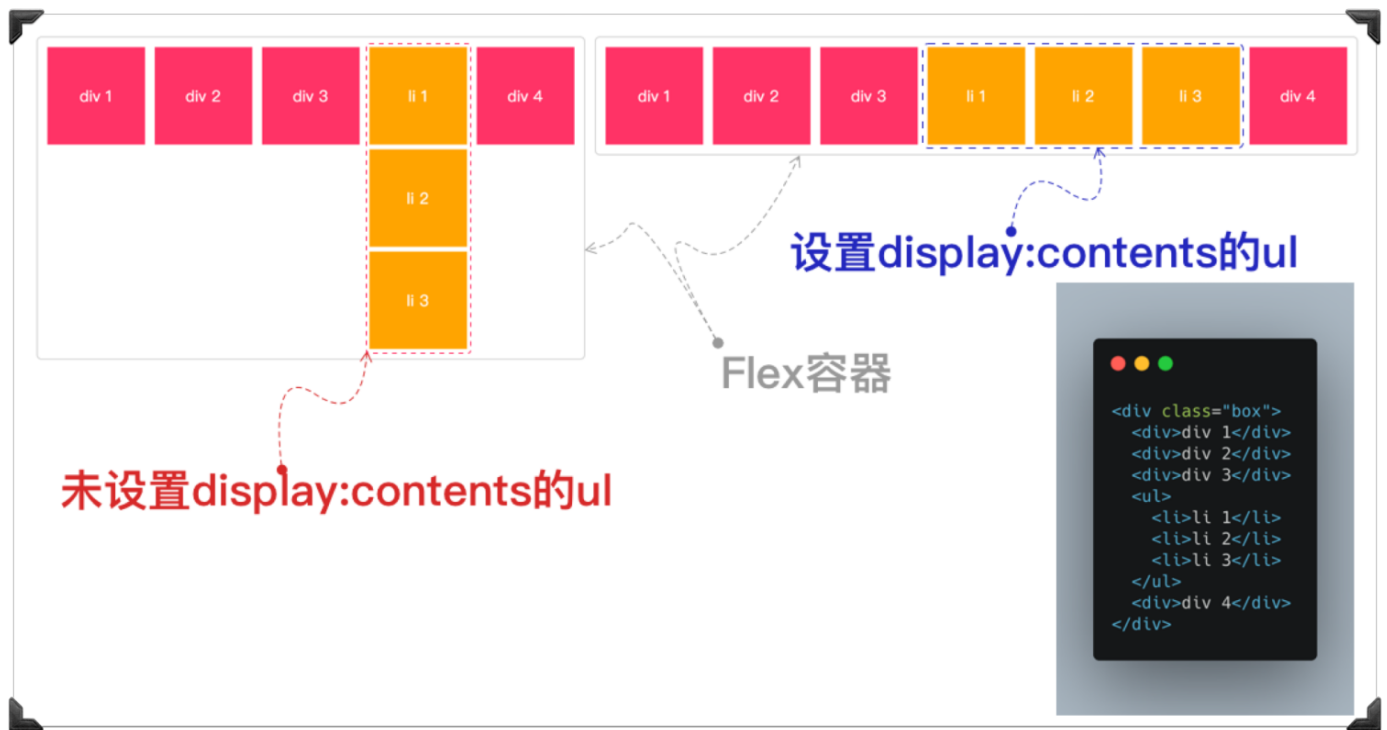
使用 `display` 对元素进行格式化之后，可以把内容装置在不同的容器中，好比我们生活中的器皿一样，不同的器皿可以看到不同的效果：



来看一个简单的示例：

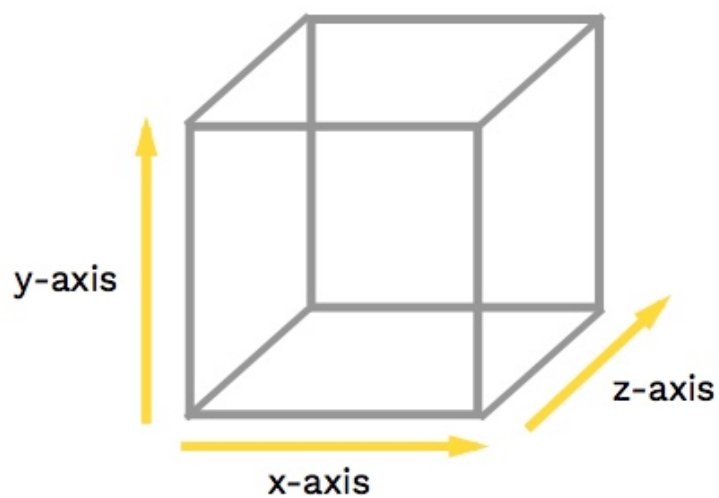


上图中最大的差异就是左侧中的 `ul` 未显式设置 `display: contents`；右侧中的 `ul` 显式设置了 `display: contents`，设置了 `display: contents` 会改变原有的格式，即其子元素 `li` 都会变成网格项目，两者效果差异如下：

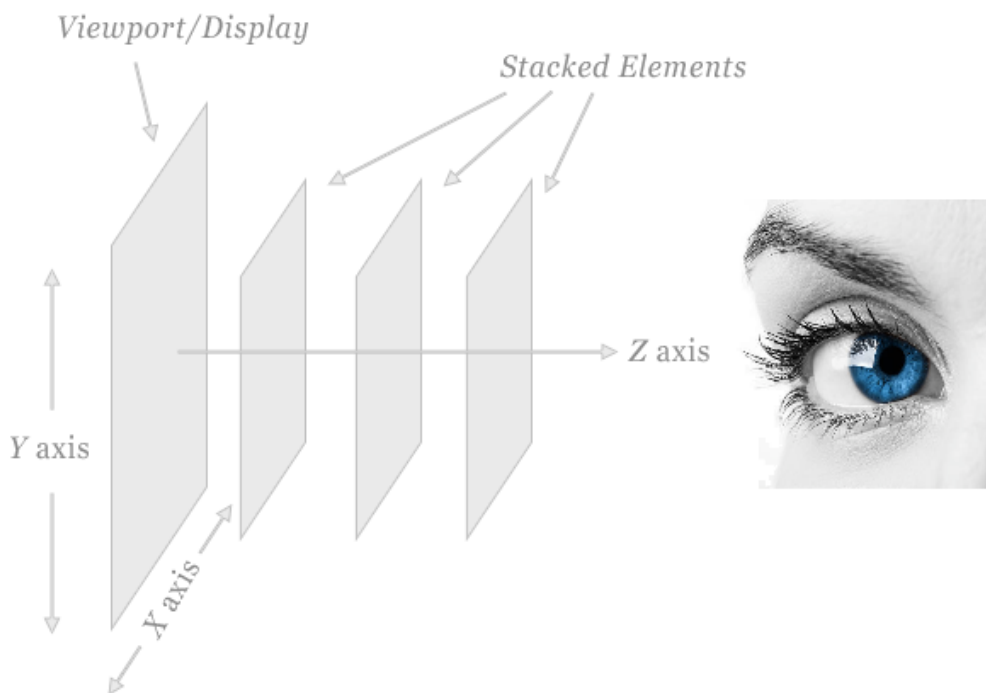


层叠上下文

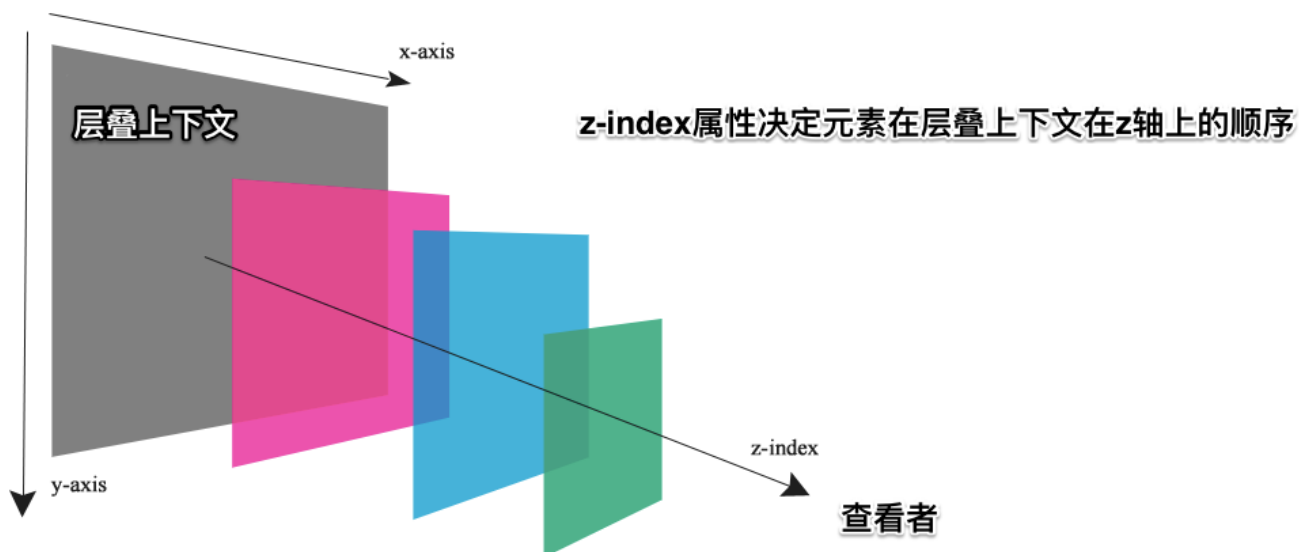
平时我们从设备终端看到的HTML文档都是一个平面的，事实上HTML文档中的元素却是存在于三个维度中。除了大家熟悉的平面画布中的 `x` 轴和 `y` 轴，还有控制第三维度的 `z` 轴。



对于 x 和 y 轴我们很易于理解，一个向右，一个向下。但对于 z 轴，理解起来就较为费力。在CSS中要确定沿着 z 轴排列元素，表示的是用户与屏幕的这条看不见的垂直线：



该系统包括一个三维 z 轴，其中的元素是层叠（Stacked）的。 z 轴的方向指向查看者， x 轴指向屏幕的右边， y 轴指向屏幕的底部。



通常，浏览器会按照CSS规范中指定的特定顺序放置元素：

在DOM树中最先出现的元素被放在首位，之后出现的元素被放在前面的元素之上。但它并不总是那么简单。只有当页面上的所有元素是自然流才起作用。也就是说，当没有元素在流中的位置被改变或者已经脱离文档流，才起作用。

事实上，每个HTML元素都属于一个层叠上下文。给定层叠上下文中的每个定位元素都具有一个整数的层叠层级，具有更大堆栈级别的元素盒子总是在具有较低堆栈级别的盒子的前面（上面）。盒子可能具有负层叠级别。层叠上下文中具有相同堆栈级别的框根据文档树出现的顺序层叠在一起。

文档中的层叠上下文由满足以下任意一个条件的元素形成：

- 根元素 (HTML)
- `z-index` 值不为 `auto` 的 `position` 值为非 `static`
- `position` 值为非 `static`
- 一个 `z-index` 值不为 `auto` 的 Flex 项目 (Flex item)，即：父元素 `display: flex|inline-flex`
- `opacity` 属性值小于 `1` 的元素
- `transform` 属性值不为 `none` 的元素
- `mix-blend-mode` 属性值不为 `normal` 的元素
- `filter`、`perspective`、`clip-path`、`mask`、`motion-path` 值不为 `none` 的元素
- `perspective` 值不为 `none` 的元素
- `isolation` 属性被设置为 `isolate` 的元素

- 在 `will-change` 中指定了任意 CSS 属性，即便你没有直接指定这些属性的值

而且每个页面都有一个默认的层叠上下文。这个层叠上下文的根就是 `html` 元素。`html` 元素中的一切都被置于这个默认的层叠上下文的一个层叠层上。

叠层水平

层叠水平（Stacking Level）决定了同一个层叠上下文中元素在z轴上的显示顺序。Level这个词很容易让我们联想到我们真正世界中的三六九等、论资排辈。在真实世界中，每个人都是独立的个体，包括双胞胎，有差异就有区分。例如，又胞胎虽然长得很像，但实际上，出生的时间还是有先后顺序的，先出生的那个就大（大哥或大姐）。网页中的元素也是如此，页面中的每个元素都是独立的个体，他们一定是会有一个类似排名排序的情况存在。而这个排名排序、论资排辈就是我们这里所说的层叠水平。层叠上下文元素的层叠水平可以理解为官员的职级，一品两品，县长省长之类；对于普通元素，这个嘛...你自己随意理解。

于是，显而易见，所有的元素都有层叠水平，包括层叠上下文元素，层叠上下文元素的层叠水平可以理解为官员的职级，一品两品，县长省长之类。然后，对于普通元素的层叠水平，我们的探讨仅仅局限在当前层叠上下文元素中。为什么呢？因为否则没有意义。

翻译成术语就是：

普通元素的层叠水平优先由层叠上下文决定，因此，层叠水平的比较只有在当前层叠上下文元素中才有意义。

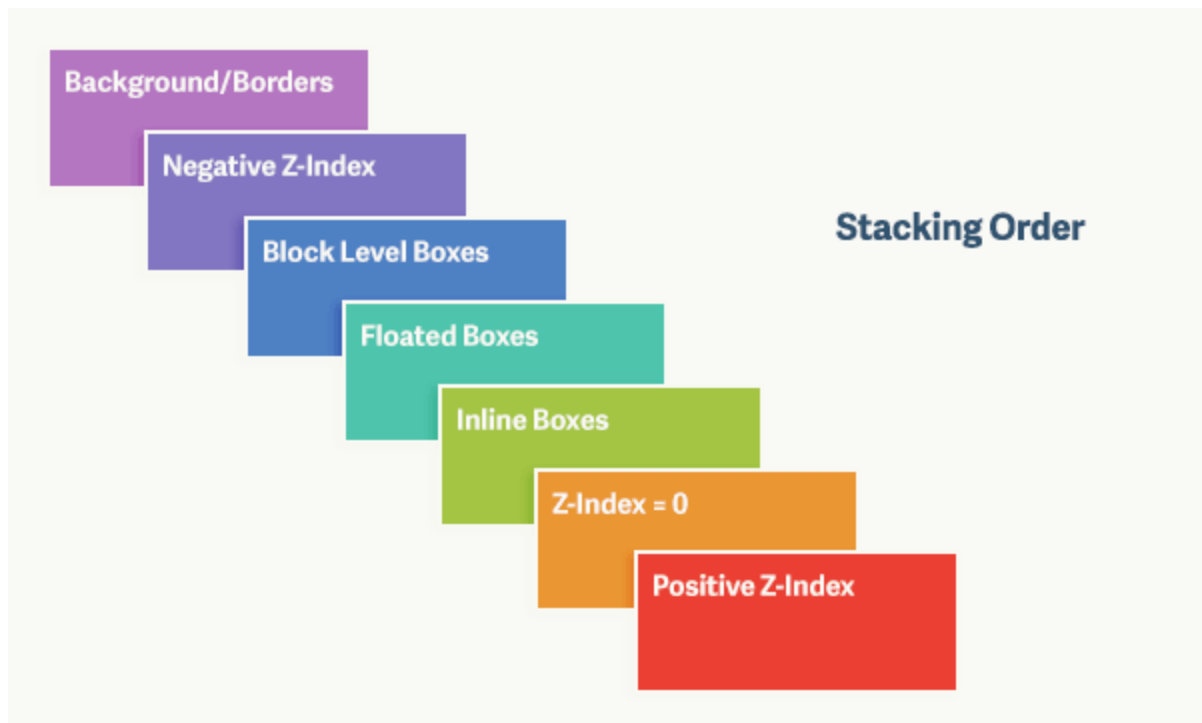
需要注意的是，诸位千万不要把层叠水平和CSS的 `z-index` 属性混为一谈。没错，某些情况下 `z-index` 确实可以影响层叠水平，但是，只限于定位元素以及Flex盒子的孩子元素；而层叠水平所有的元素都存在。

叠层顺序

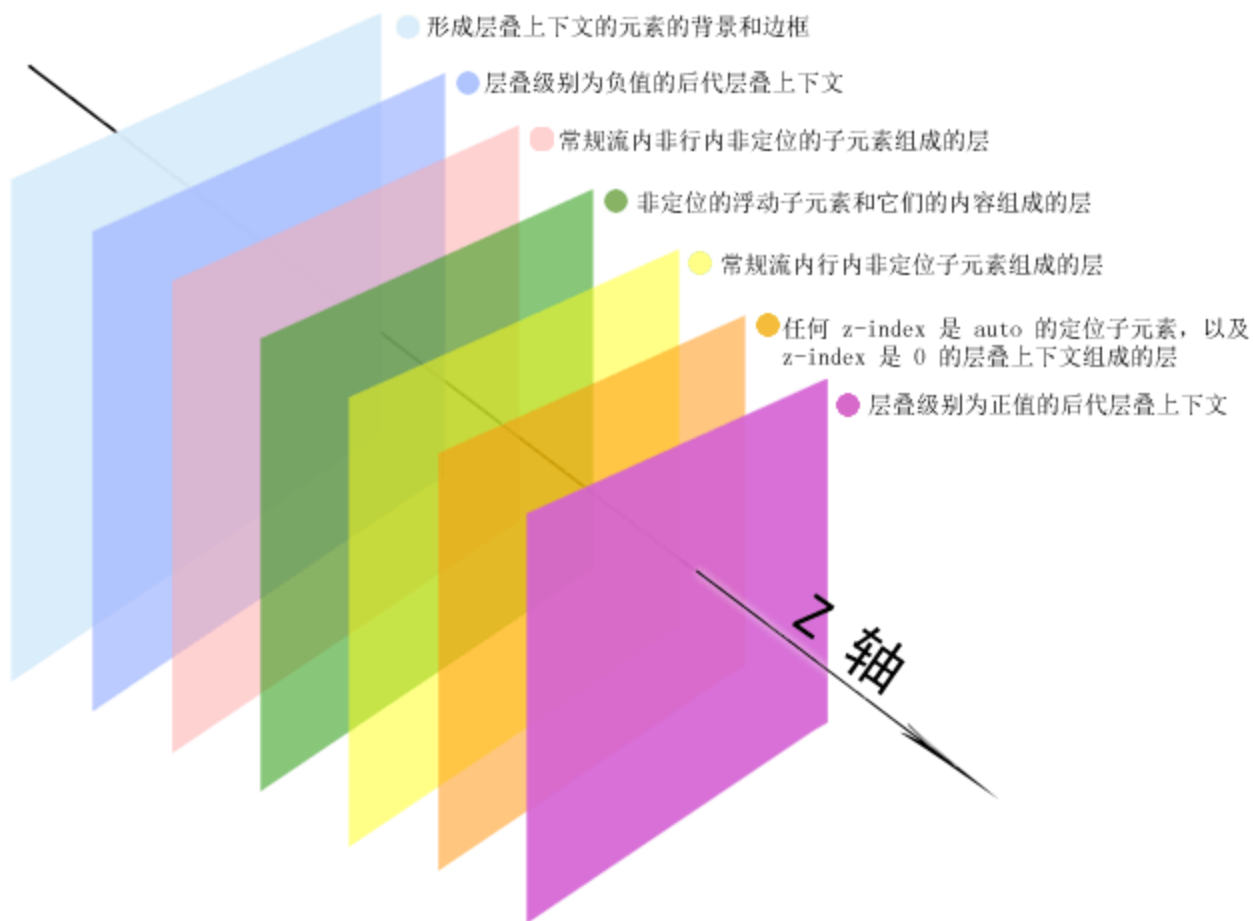
在HTML文档中，默认情况之下有一个自然层叠顺序（Natural Stacing Order），即元素在 `z` 轴上的顺序。它是由许多因素决定的。比如下面这个列表，它显示了元素盒子放入层叠顺序上下文的顺序，从层叠的底部开始，共有七种层叠等级：

- 背景和边框：形成层叠上下文的元素的背景和边框。层叠上下文中的最低等级。
- 负 `z-index` 值：层叠上下文内有着负 `z-index` 值的子元素。
- 块级盒：文档流中非行内非定位子元素。
- 浮动盒：非定位浮动元素。

- 行内盒：文档流中行内级别非定位子元素。
- `z-index: 0`：定位元素。这些元素形成了新的层叠上下文。
- 正 `z-index` 值：定位元素。层叠上下文中的最高等级。



这七个层叠等级构成了层叠次序的规则。在层叠等级七上的元素会比在等级一至六上的元素显示地更上方（更靠近观察者）。可以结合w3help中的一张图来帮助我们更好的理解这七个层叠等级：

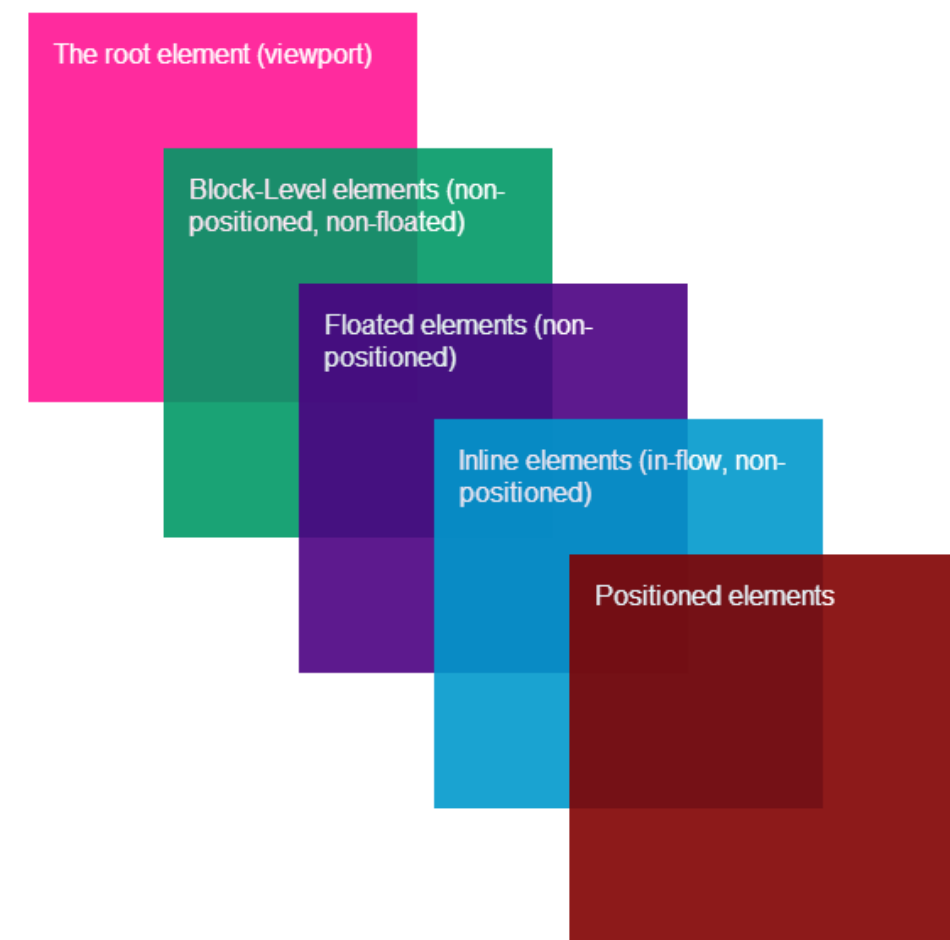


其实对于层叠顺序规则还是较为复杂的。

当页面包含浮动元素、绝对定位的元素、固定定位的元素或相对定位的元素（元素从正常位置偏移一定量）以及内联元素时，浏览器会以不同的方式显示它们（放置它们）。元素从最靠近查看者的地方排列到最远的地方，如下所示：

- 定位元素按源代码中的外观顺序排列。源代码中的最新内容最接近查看者
- 内联元素（比如文本和图像）是流入和非定位（它们的位置是静态的）
- 非浮动元素按照源代码中外观的顺序排列
- 非定位和非浮动块级元素
- 根元素 `html` 是全局层叠上下文的根，包含页面上的所有元素

这就是浏览器在呈现页面上的元素时应用的默认层叠顺序。



如果你想要更改定位元素在 `z` 轴上的渲染顺序，可以使用 `z-index` 属性。例如，你有两个绝对定位的元素，它们在某个点上重叠，并且你希望其中一个元素显示在另一个元素的前面，即使它在源代码中出现在它之前，你也可以使用 `z-index` 属性来实现这一点。

此时需要注意的第一件重要的事情是，`z-index` 属性只适用于定位元素。所以，即使为元素提供 `z-index` 的值将其置于其他元素之前，`z-index` 也不会对元素产生影响，除非它被定位；也就是说，除非它具有除 `static` 之外的 `position` 值。

因此，如果所有定位的元素具有 `z-index` 的索引值，则将元素从最靠近查看者排列到最远的位置，如下所示：

具有正值的 `z-index` 的定位元素。较高的值更接近屏幕。然后，按照它们出现在源代码中的顺序排列：

- 定位元素的 `z-index:0` 或 `z-index: auto` ；
- 内联元素（如文本和图像）是流中的和非定位的（它们的位置是静态的）
- 源代码中出现顺序的非定位浮动元素

- 非定位和非浮动块级元素
- 具有负值的 `z-index` 的定位元素。较低的 `z-index` 索引值更近。然后按照它们在源代码中出现的顺序
- 根元素 `html` 是全局层叠上下文的根，包含页面上的所有元素



当我们在定位元素上设置 `z-index` 值时，它指定该元素在它所属的层叠顺序上下文中的顺序，并且它将根据上述步骤在屏幕上渲染。

但是，当我们设置元素的 `z-index` 时会发生另一件事。获取除默认值 `auto` 之外的 `z-index` 值的元素实际上为其所有定位的后代元素创建层叠上下文。我们之前提到过，每个层叠上下文都有一个根元素，

它包含其中的所有元素。当你将 `z-index` 属性应用于这个元素时，它将在其包含的上下文中指定元素的 `z` 轴顺序，并且还将创建以该元素为根的新层叠顺序上下文。

一个具有值为 `z-index:auto` 的定位元素被视为创建了新的堆叠顺序上下文，但任何实际创建新层叠顺序上下文的定位后代和后代被视为父层叠顺序上下文的一部分，而不是新的层叠顺序上下文。

当一个元素成为一个新的层叠顺序上下文时，它所定位的后代元素将会按照我们前面提到的元素本身的规则在其中进行层叠渲染。因此，如果我们再次重写渲染过程，它会是这样的：

- 具有正值 `z-index` 的定位元素组成的层叠顺序上下文。较高的值更接近屏幕。然后按照它们在源代码中出现的顺序呈现
- 定位元素的 `z-index: 0` 或 `z-index: auto`
- 内联元素（比如文本和图像）是流中的和非定位的（它们的位置是静态的）
- 非浮动元素按照源代码中外观的顺序排列
- 非定位和非浮动块级元素
- 具有负值 `z-index` 的定位元素组成的层叠顺序上下文。较低的 `z-index` 的值更接近屏幕。然后按照它们在源代码中出现的顺序呈现
- 根元素 `html` 是全局层叠上下文的根，包含页面上所有元素

因此，当我们使用 `z-index` 属性来确定其层叠顺序中定位元素的顺序时，我们还创建了“原子（Atomic）”层叠顺序上下文，其中每个元素成为其所有定位后代的层叠顺序上下文。

样式层叠

再回过头来看样式的层叠。CSS在任何代码块中，比如 `{ }` 中都可以同时书写相同的样式规则，比如：

```
1 p {  
2     color: red;  
3     color: blue; /* 被采用 */  
4 }
```

虽然这样书写并不会引起错误，但不建议这么来写CSS样式规则。

前面我们也提到过，在同一个样式文件中也有可能在对同一个元素使用相同的样式规则，比如：

```
1 <a class="link"></a>
2
3 a {
4     color: red;
5 }
6
7 .link {
8     color: blue;
9 }
```

甚至是在不同的样式文件中也可对同一元素使用相同的样式规则，比如

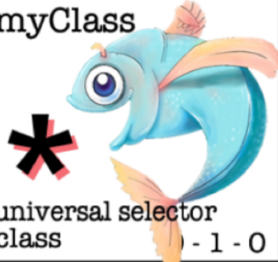
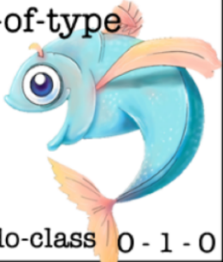
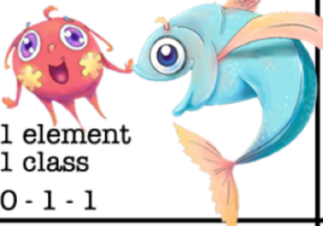
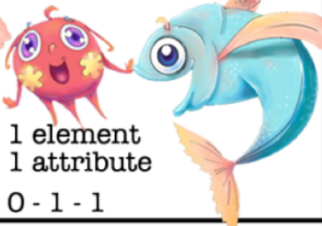
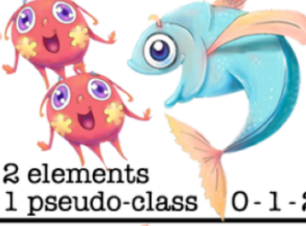
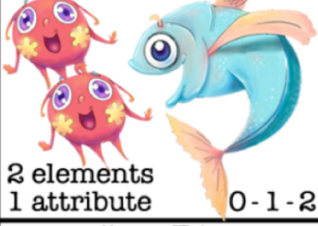
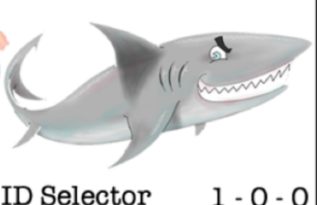
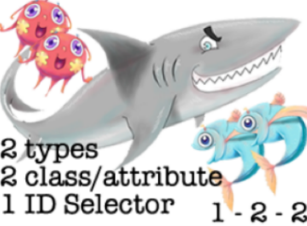
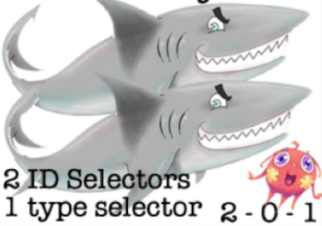
```
1 /* base.css */
2 a {
3     color: red
4 }
5
6 /* style.css */
7
8 @import base.css;
9
10 a {
11     color: orange;
12 }
13
14 <!-- index.html -->
15
16 <link href="style.css" />
17 <style>
18     a {
19         color: lime;
20     }
21 </style>
```

上面几种不同的样式写法，都可以被称为样式的层叠。只不过最终被运用到元素上的规则以最终出现的规则为准，当然这也有一个前提条件，那就是他们选择器权重是相同。

权重

在CSS领域说到权重，或许大家更多关注到的是CSS选择器的权重。其实除了选择器权重对元素样式有影响之外，也和CSS样式规则出现的先后顺序有关（正如前面所提到的层叠顺序）。而且除了选择器有权重之外，也有属性权重一说。

对于选择器权重，下图可以很形象的阐述：

*  universal selector 0 - 0 - 0	div  1 element 0 - 0 - 1	li > ul  2 elements 0 - 0 - 2	body div ... ul li p a  12 elements 0 - 0 - 12
.myClass  1 class 0 - 1 - 0	*.myClass  1 universal selector 1 class 0 - 1 - 0	[type=checkbox]  1 attribute selector 0 - 1 - 0	:only-of-type  1 pseudo-class 0 - 1 - 0
li.myClass  1 element 1 class 0 - 1 - 1	li[attr]  1 element 1 attribute 0 - 1 - 1	li:nth-of-type(3n)~li  2 elements 1 pseudo-class 0 - 1 - 2	form input[type=email]  2 elements 1 attribute 0 - 1 - 2
li.class:nth-of-type(3n)  1 element 1 class 1 pseudo-class 0 - 2 - 1	input[type]:not(.class)  1 element 1 class 1 attribute 0 - 2 - 1	cl:nth-child(odd)chk[type]...  10 class/attribute/ pseudo-classes 0 - 10 - 0	#myDiv  ID Selector 1 - 0 - 0
#myDiv li.class a[href]  2 types 2 class/attribute 1 ID Selector 1 - 2 - 2	#divitis #myDiv a  2 ID Selectors 1 type selector 2 - 0 - 1	style=""  inline style 1 - 0 - 0 - 0	!important  !important 1 - 0 - 0 - 0 - 0

选择器权重可以类似于海洋世界中的生存规则，大鱼吃小鱼，小鱼吃虾米。如果你实现不清楚两个选择器的权重，也可以使用[在线的测试工具](#)来检测：

:root

0

IDs

1

Classes, attributes and pseudo-classes

0

Elements and pseudo-elements

+ Duplicate

html

0

IDs

0

Classes, attributes and pseudo-classes

1

Elements and pseudo-elements

+ Duplicate

html.page__document

0

IDs

1

Classes, attributes and pseudo-classes

1

Elements and pseudo-elements

+ Duplicate

对于一个选择器的权重，将会按下面这样的规则进行计算：

- 如果声明来自一个行内样式（`style` 属性）而不是一条选择器样式规则，算 `1`，否则就是 `0`（`=a`）。HTML中，一个元素的 `style` 属性值是样式规则，这些属性没有选择器，所以 `a=1`，`b = 0`，`c = 0`，`d = 0`，即 `1, 0, 0, 0`
- 计算选择器中 `ID` 属性的数量（`= b`）
- 计算选择器中其它属性和伪类的数量（`= c`）
- 计算选择器中元素名和伪元素的数量（`= d`）

4个数连起来 `a-b-c-d`（在一个基数很大的数字系统中）表示特殊性，比如下面这样的示例：

```

1 *           {} /* a=0 b=0 c=0 d=0 -> 选择器权重 = 0,0,0,0 */
2 li          {} /* a=0 b=0 c=0 d=1 -> 选择器权重 = 0,0,0,1 */
3 li:first-line {} /* a=0 b=0 c=0 d=2 -> 选择器权重 = 0,0,0,2 */
4 ul li       {} /* a=0 b=0 c=0 d=2 -> 选择器权重 = 0,0,0,2 */
5 ul ol+li    {} /* a=0 b=0 c=0 d=3 -> 选择器权重 = 0,0,0,3 */
6 h1 + *[rel=up] {} /* a=0 b=0 c=1 d=1 -> 选择器权重 = 0,0,1,1 */
7 ul ol li.red {} /* a=0 b=0 c=1 d=3 -> 选择器权重 = 0,0,1,3 */
8 li.red.level {} /* a=0 b=0 c=2 d=1 -> 选择器权重 = 0,0,2,1 */
9 #x34y       {} /* a=0 b=1 c=0 d=0 -> 选择器权重 = 0,1,0,0 */
10 style=""    {} /* a=1 b=0 c=0 d=0 -> 选择器权重 = 1,0,0,0 */

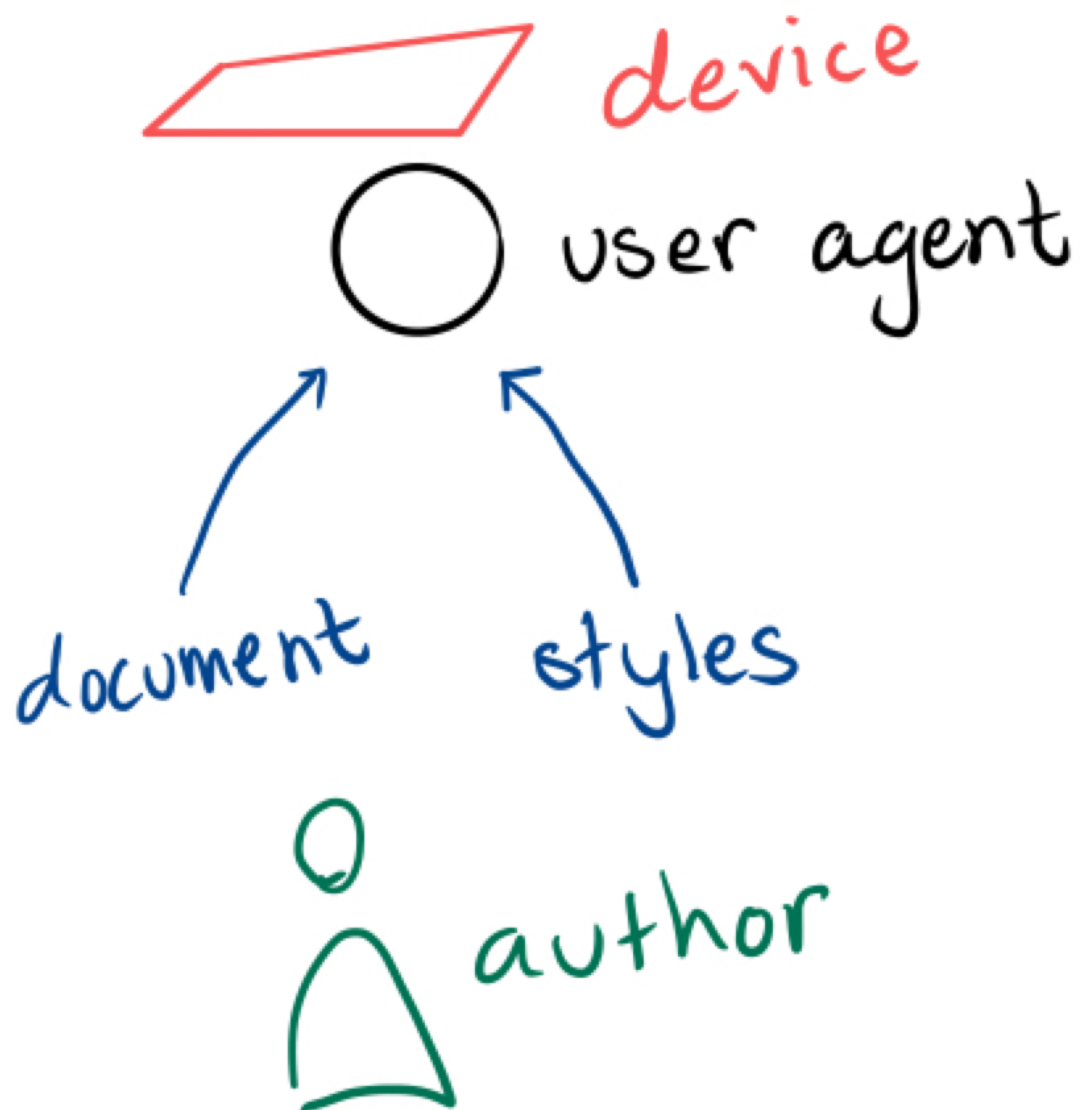
```

客户端渲染页面时，除了选择器权重会影响元素样式规则之外，还有样式来源也会影响元素样式规则。就CSS规则的来源而言，规则主要来自三个地方：

- **编写者规则（Author）**：这是HTML文档声明的CSS。也就是我们前端开发人员编写的，根据文档语言（比如HTML）约定给源文档指定样式表。这也是我们能够控制的唯一来源
- **用户（User）**：这是由浏览器的用户定义和控制的。不是每个人都会有一个，但是当人们添加一个时，通常是为了覆盖样式和增加网站的可访问性。比如，用户可以指定一个售有样式表的文件，或者用户代理可能会提供一个用来生成用户样式（或者表现得像这样做了一样）的界面
- **用户代理（User-Agent）**：这些是浏览器为元素提供的默认样式。这就是为什么 `input` 在不同的浏览器上看起来略有不同，这也是人们喜欢使用CSS重置样式，以确保重写用户代理样式的原因之一

这三种样式表将在一定范围内重叠，并且它们按照层叠互相影响。





CSS中运用于同一元素时的属性也会有权重一说，比如说同样用来描述元素宽度的属性 `width` 、 `min-width` 、 `max-width` ，比如：

with a parent of 400px

```
.block {  
  display:inline-block;  
  min-width:50%; 200px  
  max-width:100%; 400px winner  
  width:calc((480px - 100%) * 480); 38400px  
}
```

最终 `max-width: 100%` 取胜，即`max-width`权重高于 `width` 和 `min-width`，但这并不是绝对的，这和场景有着紧密关系。不过同时在这三个属性上使用了这三个属性时，他们的权重大致如下：

- 当 `width` 大于 `min-width` 时，会取 `width` 的值；当 `width` 小于 `min-width` 时，会取 `min-width` 的值
- 当 `width` 大于 `max-width` 时，会取 `max-width` 的值；当 `width` 小于 `max-width` 时，会取 `width` 的值
- 如果 `min-width` 大于 `max-width`，那么 `min-width` 值将作为元素的宽度；如果 `min-width` 小于 `max-width`，那么 `min-width` 值将作为元素的宽度

有关于这方面更详细的介绍可以阅读《[图解CSS: 元素尺寸的设置](#)》一文。

如果你使用Flexbox来构建Web的布局，那么对于Flex项目的宽度也有一定的权重规则：

首先根据 `content` → `width` → `flex-basis` 来决定用哪个来决定用于Flex项目。如果Flex项目显式设置了 `flex-basis` 属性，则会忽略 `content` 和 `width`。而且 `min-width` 是用来设置Flex项目的下限值；`max-width` 是用来设置Flex项目的上限值。

即：

- `flex-basis` 大于 `max-width`，Flex项目的宽度等于 `max-width`，即 `max-width` 能覆盖 `flex-basis`
- 如果 `flex-basis` 小于 `min-width` 时，Flex项目的宽度会取值 `min-width`，即 `min-width` 覆盖 `flex-basis`

由于 `min-width` 大于 `max-width` 时会取 `min-width`，有了这个先取条件我们就可以将 `flex-basis` 和 `min-width` 做权重比较，即：`flex-basis` 会取于 `min-width`。反过来，如果 `min-`

`width` 小于 `max-width` 时则依旧会取 `max-width`，同时要是 `flex-basis` 大于 `max-width` 就会取 `max-width`。

注意，Flexbox布局中有关于Flex项目的宽度计算是非常复杂的，如果你对这方面知识感兴趣的话，可以阅读《你真的了解CSS的 `flex-basis` 吗？》一文。

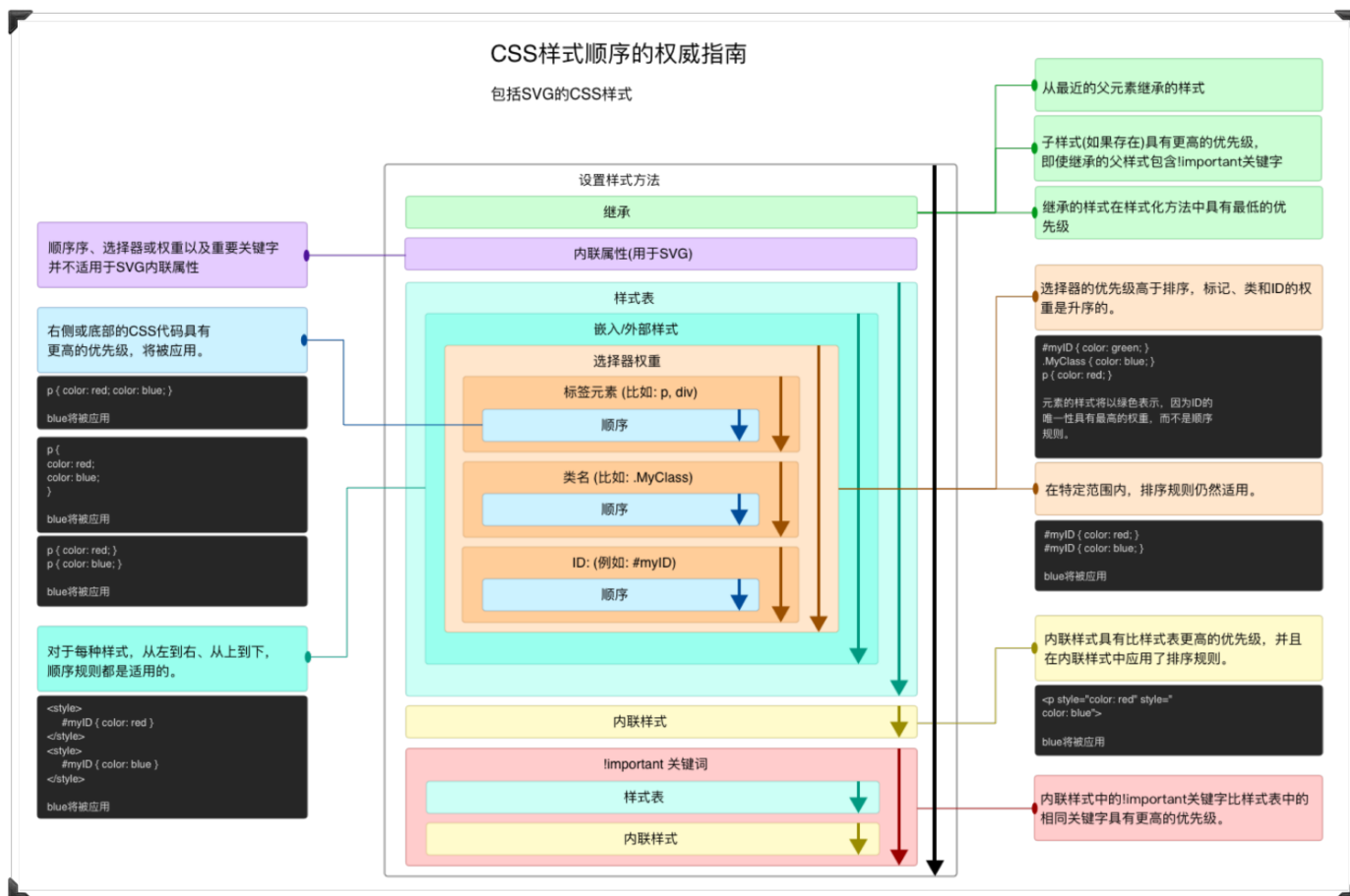
继承

“继承”是CSS中另一个重要概念。在W3C规范中，描述每个CSS属性时都会有一个选项是“Inherited”，如果值为“no”表示该属性是不可继承的，比如下图左侧的 `border` 属性；如果值为“yes”表示该属性是可继承的，比如下图右侧的 `color` 属性：

Name:	<code>'border'</code>	Name:	<code>'font-size'</code>
Value:	<code><line-width> <line-style> <color></code>	Value:	<code><absolute-size> <relative-size> <length-percentage></code>
Initial:	See individual properties	Initial:	medium
Applies to:	all elements	Applies to:	all elements
Inherited:	no	Inherited:	yes
Percentages:	N/A	Percentages:	refer to parent element's font size
Media:	visual	Media:	visual
Computed value:	see individual properties	Computed value:	an absolute length, as clamped by <code>'font-min-size'</code> and <code>'font-max-size'</code>
Animatable:	see individual properties	Canonical order:	per grammar

在CSS中提供了处理CSS继承的机制，简单地讲就是CSS提供了几个属性值，可以用来处理属性的继承。这几个属性值就是 `initial`、`inherit`、`unset` 和 `revert`。其实除了这四个属性值之外，还有一个 `all` 属性值。虽然这几个属性值主要用来帮助大家处理CSS属性继承的，但他们之间的使用，还是有一定的差异化。详细的介绍可以阅读《图解CSS：CSS层叠和继承》一文。

也就是说，CSS的层叠、权重和继承都会对你的CSS有影响。在实际使用的时候，如果很好的运用这些概念和手段，可以更好的帮助大家少写很多样式代码，而且更易于维护自己的CSS代码。



这里提到的CSS层叠、权重和继承都是CSS的基础概念，也是CSS的核心概念，掌握或者了解这些概念对于每一位从事Web开发者同学而言都是很有必要的。只有掌握这些概念，有助于帮助大家更好的理解CSS和正确的使用CSS。而这些概念在一篇文章中是无法很清晰的向大家阐述清楚。所以建议大家抽空阅读下面这几篇文章：

- [聊聊CSS中的层叠相关概念](#)
- [CSS定位和层叠控制](#)
- [CSS层叠和继承](#)
- [管理CSS层叠](#)
- `display` 属性

视觉格式化模型

首先要声明一点：

视觉格式化模型和CSS盒模型不是同一个东西！

简单点说呢。Web页面（文档树）是由很一个个盒子组成（因为任何元素都可以被视为是一个盒子），而视觉格式化模型却是一套规则，用来计算元素转换为盒子的规则。而页面的布局都由这些盒子的所处的各处位置组合而成。那么理解了元素怎么转成盒子的规则，就理解了Web页面是怎么布局。而每个盒子的布局主要由以下几个因素决定：

- **盒子的尺寸**：精确指定、由约束条件指定或没有指定
- **盒子的类型**：行内盒子（inline）、行内级盒子（inline-level）、原子行内级盒子（atomic inline-level）和块盒子（block）
- **定位方案**：普通流定位、浮动定位或绝对定位
- **文档树中的其它元素**：即当前盒子的子元素或兄弟元素
- **视窗尺寸与位置**
- **所包含的图片的尺寸**
- **其他的某些外部因素**

如果你想彻底理解CSS的视觉可式化模型，其中还有一些概念需要掌握，比如：

- **块（Block）**：一个抽象的概念，一个块在文档流上占据一个独立的区域，块与块之间在垂直方向按照顺序依次堆叠（默认情况之下）
- **包含块（Containing Block）**：指的是包含其他盒子的块
- **盒子（Box）**：一个抽象的概念，由CSS引擎根据文档中的内容所创建，主要用于文档元素的定位、布局和格式化等用途
- **块级盒子（Block-level Box）**：由块级元素生成。一个块级元素至少会生成一个块级盒子，但也有可能生成多个（例如列表项元素）
- **行内级盒子（Inline-level Box）**：由行内级元素生成。同样的，一个行内元素至少会生成一个行内级盒子。行内级盒子包括行内盒子和原子行内级盒子两种，区别在于该盒子是否参与行内格式化上下文的创建
- **块盒子（Block Box）**：如果一个块级盒子同时也是一个块容器盒子，则称其为块盒子。除具名块盒子之外，还有一类块盒子是匿名的，称为匿名块盒子（Anonymous Block Box），匿名盒子无法被CSS选择器选中
- **原子行内级盒子（Atomic Inline-level Box）**：不参与行内格式化上下文创建的行内级盒子。原子行内级盒子一开始叫做原子行内盒子（Atomic Inline Box），后被修正。原子内级盒子的内容不会拆分成多行显示。
- **行盒（Line Box）**：和行内盒是不一样的。行盒是由行内格式化上下文（Inline Formatting Context）产生的盒子，用于表示一行。行盒从包含块的一边排版到另一边。一般情况下，浏览器会为每行创建一个看不见的行盒
- **块容器盒子（Block Container Box或Block Containning Box）**：块容器盒子侧重于当前盒子作为容器角色，它不参与当前块的布局和定位，它所描述的仅仅是当前盒子与其后代之间的关系
- **块级元素（Block-level Element）**：是指元素的 `display` 值为 `block`、`list-item`、`table`、`flex` 和 `grid` 等，该元素将成为块级元素。元素是否是块级元素仅是元素本身的属性，

并不直接用于格式化上下文的创建或布局

- **行内级元素 (Inline-level Element)**：是指元素的 `display` 值为 `inline`、`inline-block`、`inline-table`、`inline-flex` 和 `inline-grid` 等，该元素将成为行内级元素。与块元素一样，元素是否是行内级元素仅是元素本身的属性，并不直接用于格式化上下文的创建或布局
- **视窗 (Viewport)**：视窗就是浏览器中可视区域。视窗大小指的就是浏览器可视区域的大小
- **匿名盒子**：分为块匿名盒子和行内匿名盒子。在某些情况下进行视觉格式化时，需要添加一些增补性的盒子，这些盒子无法被CSS的选择器选中，而这种盒子被称为匿名盒子 (Anonymous Box)。

除了上述说到的盒子，在CSS中还定义了几种内容模型，这些模型同样可以应用于元素。这些模型一般用来描述布局，它们可能会定义一些额外的盒子类型：

- **表格内容模型**：可能会创建一个表格包装器盒子和一个表格盒子，以及多个其他盒子如表格标题盒子等
- **多列内容模型**：可能会在容器盒子和内容之间创建多个列盒子
- **Flexbox内容模型**：可能会创建一个弹性盒子
- **Grid内容模型**：可能会创建一个网格盒子

就这些概念也足让我们感到烦人了吧。我想你看到这些概念，应该不会再说CSS容易了。

有了这些概念，我们再来说CSS中的格式化上下文。我想你或多或少听过这个词吧。在CSS中，格式化上下文有很多种，除了大家熟悉的 **BFC**、**IFC** 之外还有由Flexbox布局创建的 **FFC** 和Grid布局创建 **GFC** 等。这些统称为CSS **格式化上下文**，也被称作 **视觉格式化模型**。而CSS视觉格式化模型是用来处理文档并将它显示在视觉媒体上的机制。

简单地说，就是用来控制盒子的位置，即实现页面的布局。

格式化上下文也可以说是CSS视觉渲染中的一部分，其主要作用是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。那么理解CSS格式化上下文有助于我们掌握各类CSS布局的关键。

行内格式化上下文

行内格式化上下文 (Inline Formatting Context)，简称 **IFC**。主要用来规则行内级盒子的格式化规则。

IFC的行盒的高度是根据包含行内元素中最高的实际高度计算而来。主要会涉及到CSS中的 `font-size`、`line-height`、`vertical-align` 和 `text-align` 等属性。

行内元素从包含块顶端水平方向上逐一排列，水平方向上的 `margin`、`border`、`padding` 生效。行内元素在垂直方向上可按照顶部、底部或基线对其。

当几个行内元素不能在一个单独的行盒中水平放置时，他们会被分配给两个或更多的(Vertically-stacked Line Box)垂直栈上的行盒，因此，一个段落是很多行盒的垂直栈。这些行盒不会在垂直方向上被分离（除非在其他地方有特殊规定），并且他们也不重叠。

- 垂直方向上，当行内元素的高度比行盒要低，那么 `vertical-align` 属性决定垂直方向上的对齐方式。
- 水平方向上，当行内元素的总宽度比行盒要小，那么行内元素在水平方向上的分部由 `text-align` 决定。
- 水平方向上，当行内元素的总宽度超过了行盒，那么行内元素会被分配到多个行盒中去，如果设置了不可折行等属性，那么行内元素会溢出行盒。
- 行盒的左右两边都会触碰到包含块，而 `float` 元素则会被放置在行盒和包含块边缘的中间位置。

下面这些规则都会创建一个行内格式化上下文：

- IFC只有在一个块级元素中仅包含行内级元素时才会生成
- 内部的盒子会在不平方向，一个接一个的放置
- 这些盒子垂直方向的起点从包含块盒子的顶部开始
- 摆放这些盒子的时候，它们在水平方向上的 `padding`、`border` 和 `margin` 所占用的空间都会被考虑在内
- 在垂直方向上，这些盒子可能会以不同形式来对齐（`vertical-align`）
- 能把在一行上的盒子都完全包含在一行行盒（Line Box），行盒的宽度是由包含块和存在的浮动来决定
- IFC中的行盒一般左右边都紧贴其包含块，但是会因浮动元素的存在发生变化。浮动元素会位于IFC与行盒之间，使得行盒宽度缩短
- 当行内级盒的总宽度小于包含它们的行盒时，其水平渲染规则则由 `text-align` 来确定
- 当行内盒超过行盒的宽度时，它会被分割成多个盒子，这些盒子被分布在多个行盒里。如果一个行内盒不能被分割，则会溢出行盒

IFC主要用于：

- 行内元素按照 `text-align` 进行水平居中
- 行内元素撑开父元素高度，通过 `vertical-align` 属性进行垂直居中

块格式化上下文

块格式化上下文（Block Formatting Context, BFC）是Web页面的可视化CSS渲染的一部分，是块盒子的布局过程发生的区域，也是浮动元素与其他元素交互的区域。

BFC实际上就是页面中一块渲染区域，该区域与其他区域隔离开来。容器里面子元素不会影响到外部，外部的元素也不会影响到容器里的子元素。

BFC 内部的盒子会从上至下一个接着一个顺序排列。BFC 内的垂直方向的盒子距离以 `margin` 属性为准，上下 `margin` 会叠加。每个元素的左侧最外层边界与包含块 BFC 的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。BFC 的区域不会与浮动元素的盒子折叠。BFC 的高度也会受到浮动元素的影响，浮动元素参与计算。

下面这些规则可以创建一个BFC：

- 根元素或包含根元素的元素
- 浮动元素（元素的 `float` 不是 `none`）
- 绝对定位元素（元素的 `position` 为 `absolute` 或 `fixed`）
- 行内块元素（元素的 `display` 为 `inline-block`）
- 表格单元格（元素的 `display` 为 `table-cell`，HTML表格单元格默认为该值）
- 表格标题（元素的 `display` 为 `table-caption`，HTML表格标题默认为该值）
- 匿名表格单元格元素（元素的 `display` 为 `table`、`table-row`、`table-row-group`、`table-header-group`、`table-footer-group`（分别是HTML `table`、`row`、`tbody`、`thead`、`tfoot` 的默认属性）或 `inline-table`）
- `overflow` 值不为 `visible` 的块元素
- `display` 值为 `flow-root` 的元素
- `contain` 值为 `layout`、`content` 或 `strict` 的元素
- 弹性元素（`display` 为 `flex` 或 `inline-flex` 元素的直接子元素）
- 网格元素（`display` 为 `grid` 或 `inline-grid` 元素的直接子元素）
- 多列容器（元素的 `column-count` 或 `column-width` 不为 `auto`，包括 `column-count` 为 `1`）
- `column-span` 为 `all` 的元素始终会创建一个新的BFC，即使该元素没有包裹在一个多列容器中

块格式化上下文包含创建它的元素内部的所有内容。其主要使用：

- 创建独立的渲染环境
- 防止因浮动导致的高度塌陷
- 防止上下相邻的外边距折叠

Flex格式化上下文

Flex格式化上下文（Flexbox Formatting Context）俗称 **FFC**。当 `display` 取值为 `flex` 或 `inline-flex`，将会创建一个Flexbox容器。该容器为其内容创建一个新的格式化上下文，即Flex格式

化上下文。

不过要注意的是，Flexbox容器不是块容器（块级盒子），下列适用于块布局的属性并不适用于Flexbox布局：

- 多列中的 `column-*` 属性不适用于Flexbox容器
- `float` 和 `clear` 属性作用于Flex项目上将无效，也不会把让Flex项目脱离文档流
- `vertical-align` 属性作用于Flex项目上将无效
- `::first-line` 和 `::first-letter` 伪元素不适用于Flexbox容器，而且Flexbox容器不为他们祖先提供第一个格式化的行或第一个字母

Grid格式化上下文

Grid格式化上下文（Grid Formatting Context），俗称 **GFC**。和FFC有点类似，元素的 `display` 值为 `grid` 或 `inline-grid` 时，将会创建一个Grid容器。该容器为其内容创建一个新的格式化上下文，即Grid格式化上下文。这和创建BFC是一样的，只是使用了网格布局而不是块布局。

网格容器不是块容器，因此一些假定为块布局设计的属性并不适用于网格格式化上下文中。特别是：

- `float` 和 `clear` 运用于网格项目将不会生效。但是 `float` 属性仍然影响网格容器子元素上 `display` 的计算值，因为这发生在确定网格项目之前
- `vertical-align` 运用于网格项目也将不会生效
- `::first-line` 和 `::first-letter` 伪元素不适用于网格容器，而且网格容器不向它们祖先提供第一个格式化行或第一个格式化字母

对于视觉格式化模型更详细的介绍可以阅读《[Web布局：视觉格式化模型](#)》一文。

盒模型

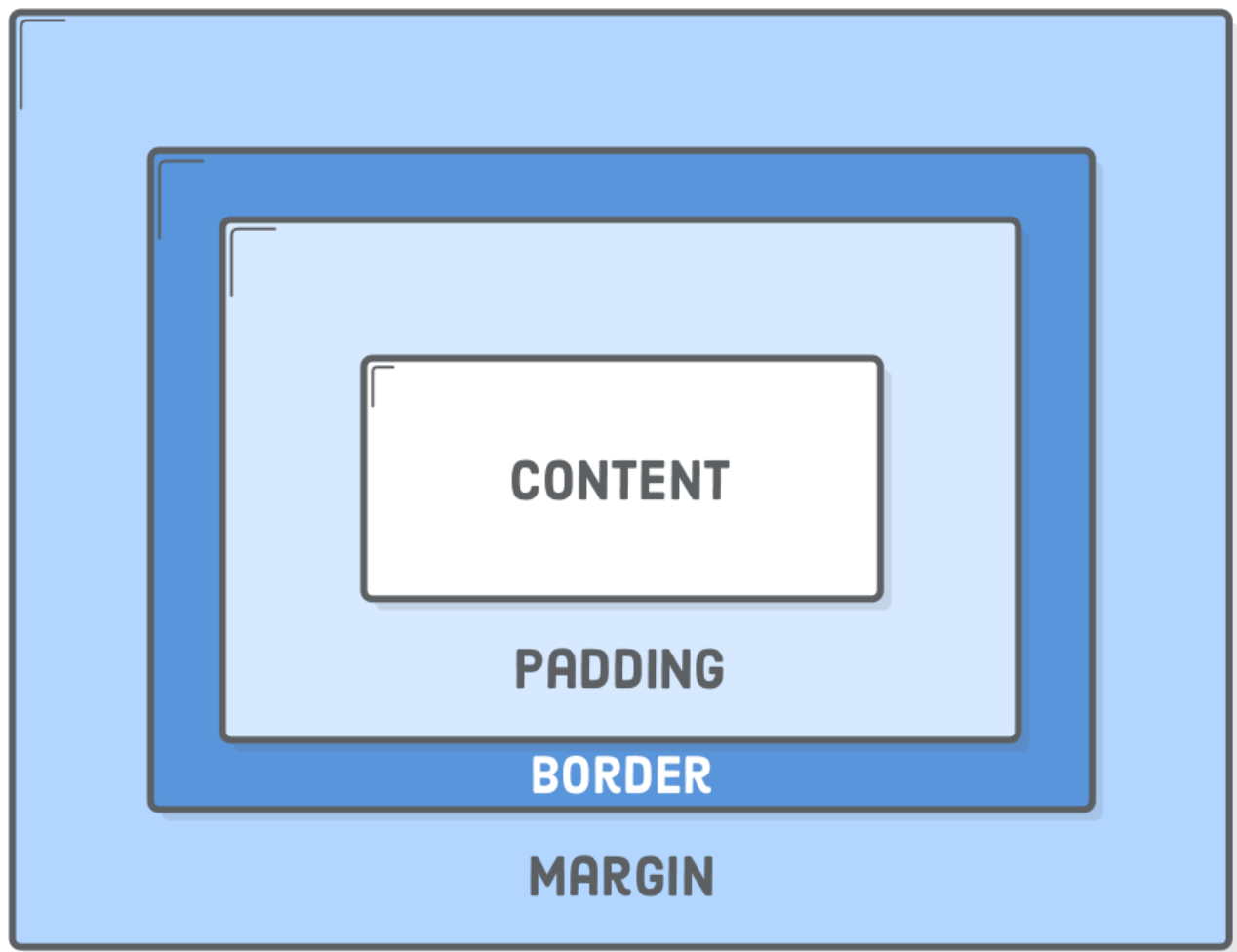
在CSS世界中，任何一个HTML元素可以解析成一个盒子。可以通过CSS属性（物理属性）来决定盒子的大小。而决定盒子大小主要由四个属性来决定：

- **Content**：元素中的文本、图像或其他媒体内容
- **padding**：盒子内容格边框之间的空间
- **border**：盒子的边框
- **margin**：盒子与其他盒子之间的距离

如果我们用一张图来描述的话，可以类似下面这样的：



上面提到的几个属性就是浏览器渲染元素盒子所需要的一切。对于CSSer来说，都喜欢用类似下图来阐述元素的盒模型：



而在浏览器调试器中“Computed”可以看到它是怎么来解释一个元素的盒模型：

position 0

margin -

border -

padding -

325 x 257.984

0 0

0

Filter

- box-sizing border-box
- color rgb(93, 96, 99)
- display block
- font-family Century
- font-size 17px
- height 257.984px
- line-height 27.2px
- margin-bottom 0px
- margin-left 0px
- margin-right 0px
- margin-top 0px
- padding-bottom 0px
- padding-left 0px
- padding-right 0px
- padding-top 0px
- position absolute
- width 325px
- webkit-font-smoothing antialiased

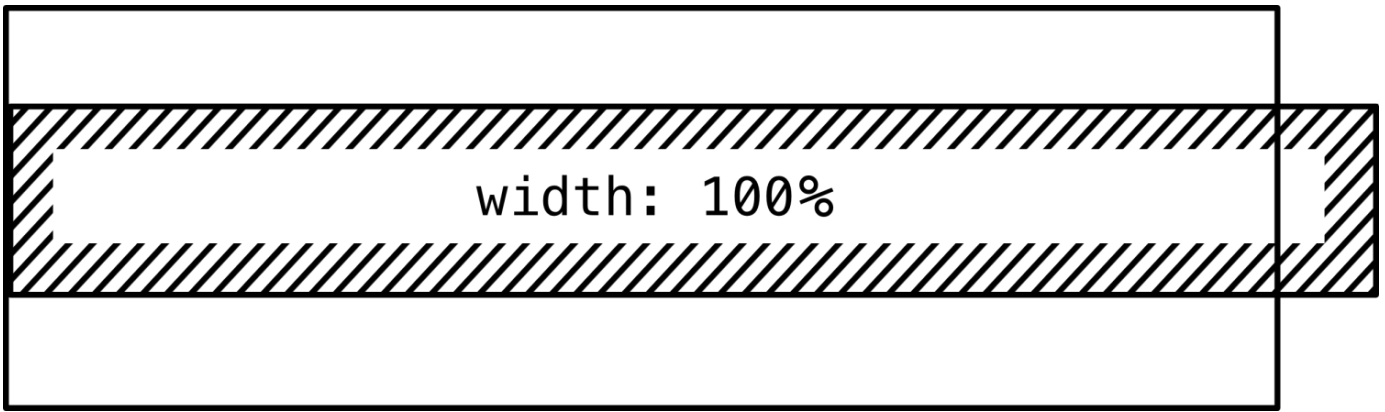
在一般情况之下，我们所说的盒子的 `width` 是元素内容的宽度，内距和边框的和，这也常常被称之为内盒的宽度。如果你在内盒的宽度上加外距的大小就可以计算出盒子外盒的宽度。

盒子的 `height` 计算和 `width` 类似。

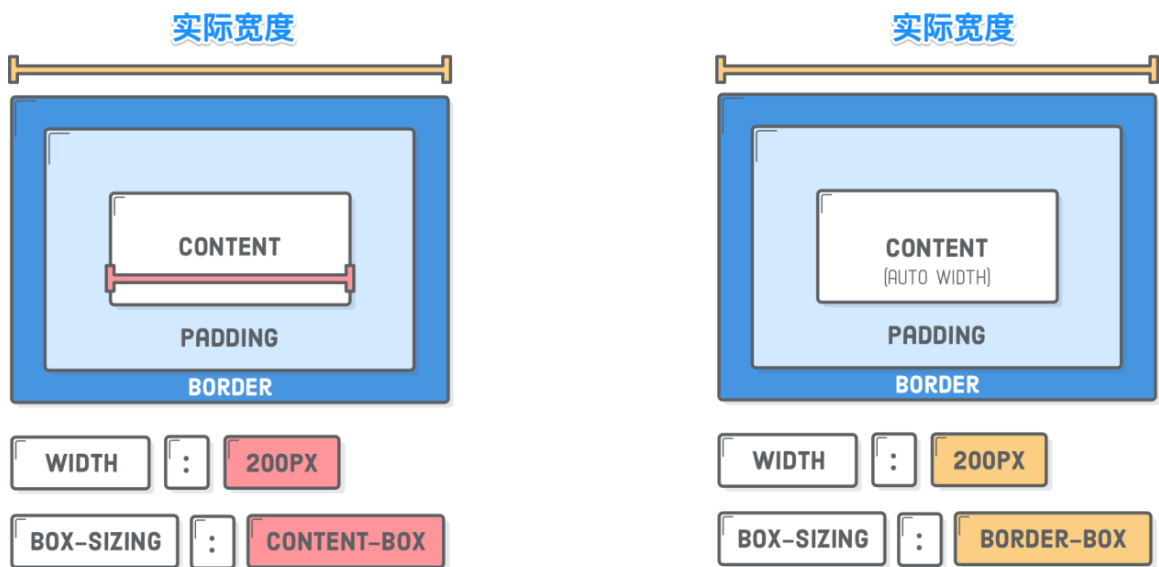
但在布局中，盒子的宽度计算会直接影响到布局，甚至会直接打破页面的布局。比如说，一个 `div` 元素：

```
1 div {  
2   width: 100%;  
3   padding: 1rem;  
4 }
```

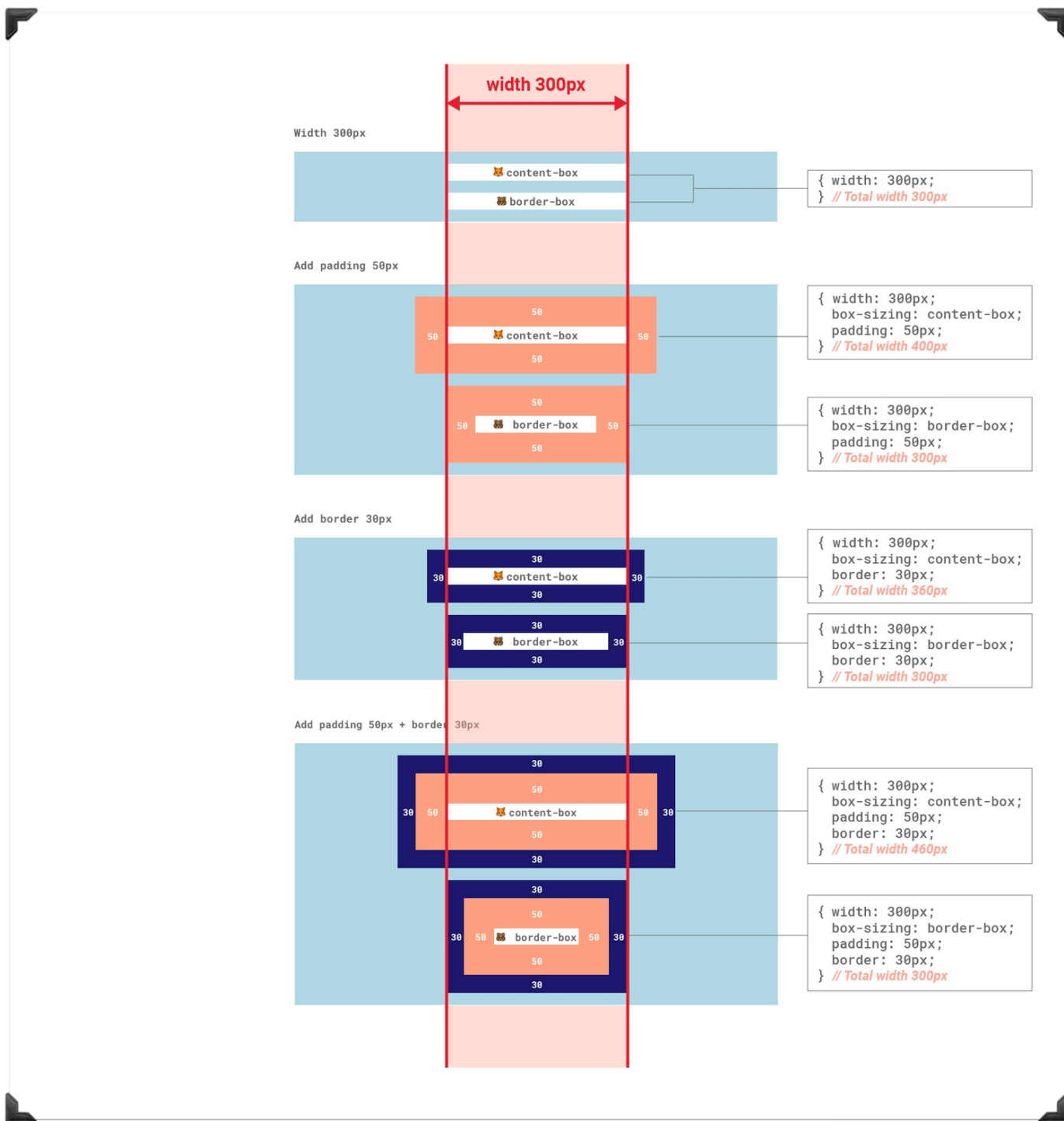
会让 `div` 超出容器，类似下图这样：



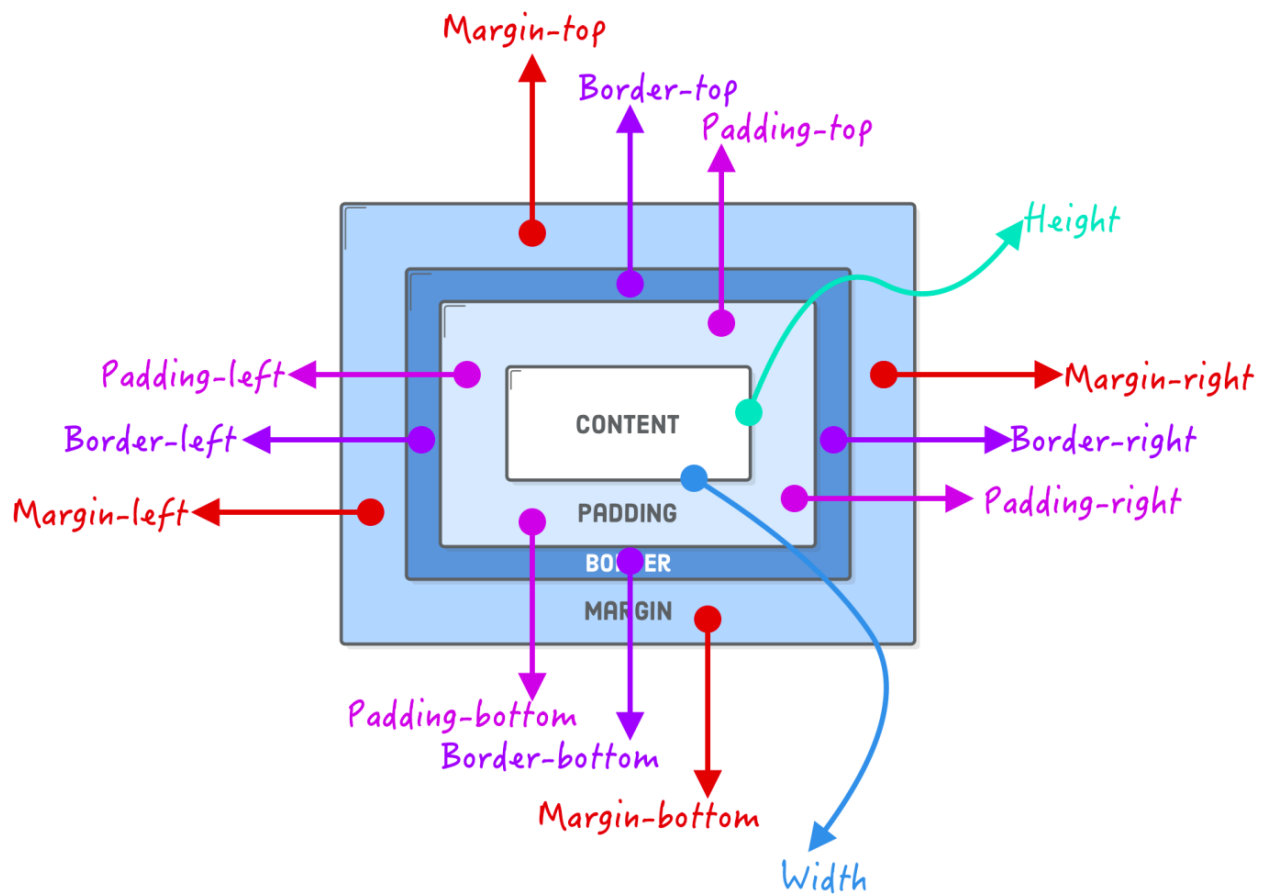
面对这样的场景时，就需要借助CSS的 `box-sizing` 属性，它可以更好的帮助我们控制盒子的大小：



用个实例来解释，这样更易于理解：



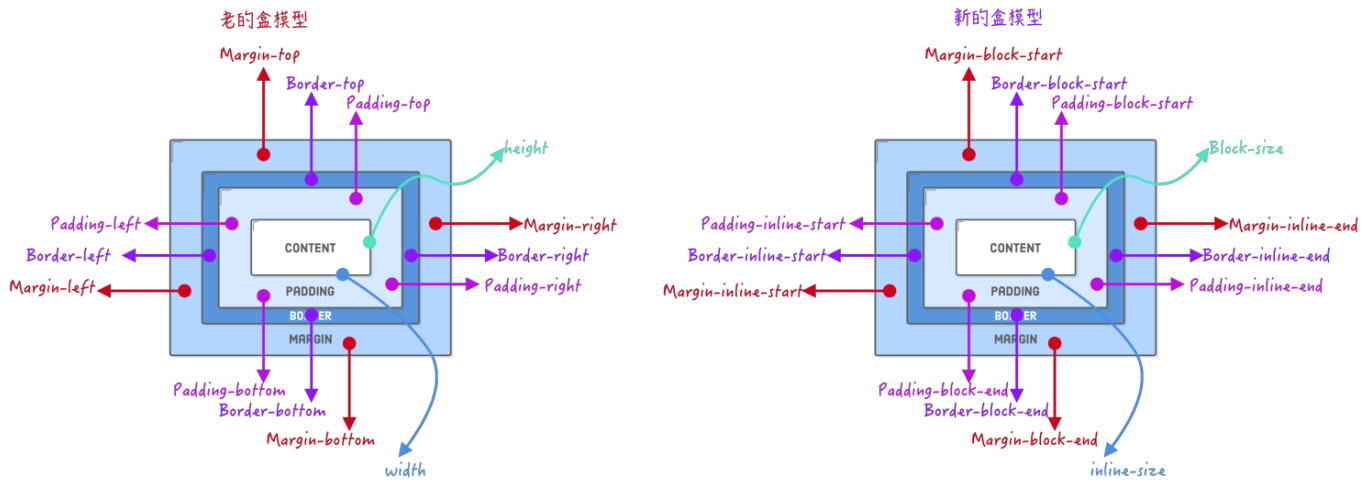
大家是否有留意到，在前面提到的 `padding`、`border` 和 `margin` 等都是采用的物理特性来描述一个盒子，而且开发者讨论盒模型的时候，都习惯使用下图来阐述它：



但随着CSS的逻辑属性的出现，在CSS中就除我们所熟悉的物理属性之外，还新增了很多逻辑属性：

物理属性	逻辑属性					
writing-mode	horizontal-tb		vertical-lr		vertical-rl	
direction	ltr	rtl	ltr	rtl	ltr	rtl
width	inline-size		block-size		block-size	
height	block-size		inline-size		inline-size	
min-width	min-inline-size		min-block-size		min-block-size	
min-height	min-block-size		min-inline-size		min-inline-size	
max-width	max-inline-size		max-block-size		max-block-size	
max-height	max-block-size		max-inline-size		max-inline-size	
border-top	border-block-start	border-block-start	border-inline-start	border-inline-end	border-inline-start	border-inline-end
border-right	border-inline-end	border-inline-start	border-block-end	border-block-end	border-block-start	border-block-start
border-bottom	border-block-end	border-block-end	border-inline-end	border-inline-start	border-inline-end	border-inline-start
border-left	border-inline-start	border-inline-end	border-block-start	border-block-start	border-block-end	border-block-end
padding-top	padding-block-start	padding-block-start	padding-inline-start	padding-inline-end	padding-inline-start	padding-inline-end
padding-right	padding-inline-end	padding-inline-start	padding-block-end	padding-block-end	padding-block-start	padding-block-start
padding-bottom	padding-block-end	padding-block-end	padding-inline-end	padding-inline-start	padding-inline-end	padding-inline-start
padding-left	padding-inline-start	padding-inline-end	padding-block-start	padding-block-start	padding-block-end	padding-block-end
margin-top	margin-block-start	margin-block-start	margin-inline-start	margin-inline-end	margin-inline-start	margin-inline-end
margin-right	margin-inline-end	margin-inline-start	margin-block-end	margin-block-end	margin-block-start	margin-block-start
margin-bottom	margin-block-end	margin-block-end	margin-inline-end	margin-inline-start	margin-inline-end	margin-inline-start
margin-left	margin-inline-start	margin-inline-end	margin-block-start	margin-block-start	margin-block-end	margin-block-end
border-top-left-radius	border-start-start-radius	border-start-end-radius	border-start-start-radius	border-start-end-radius	border-end-start-radius	border-end-end-radius
border-top-right-radius	border-start-end-radius	border-start-start-radius	border-end-start-radius	border-end-end-radius	border-start-start-radius	border-start-end-radius
border-bottom-right-radius	border-end-end-radius	border-end-start-radius	border-end-end-radius	border-end-start-radius	border-start-end-radius	border-start-start-radius
border-bottom-left-radius	border-end-start-radius	border-end-end-radius	border-start-end-radius	border-start-start-radius	border-end-end-radius	border-end-start-radius
top	inset-block-start	inset-block-start	inset-inline-start	inset-inline-end	inset-inline-start	inset-inline-end
right	inset-inline-end	inset-inline-start	inset-block-end	inset-block-end	inset-block-start	inset-block-start
bottom	inset-block-end	inset-block-end	inset-inline-end	inset-inline-start	inset-inline-end	inset-inline-start
left	inset-inline-start	inset-inline-end	inset-block-start	inset-block-start	inset-block-end	inset-block-end

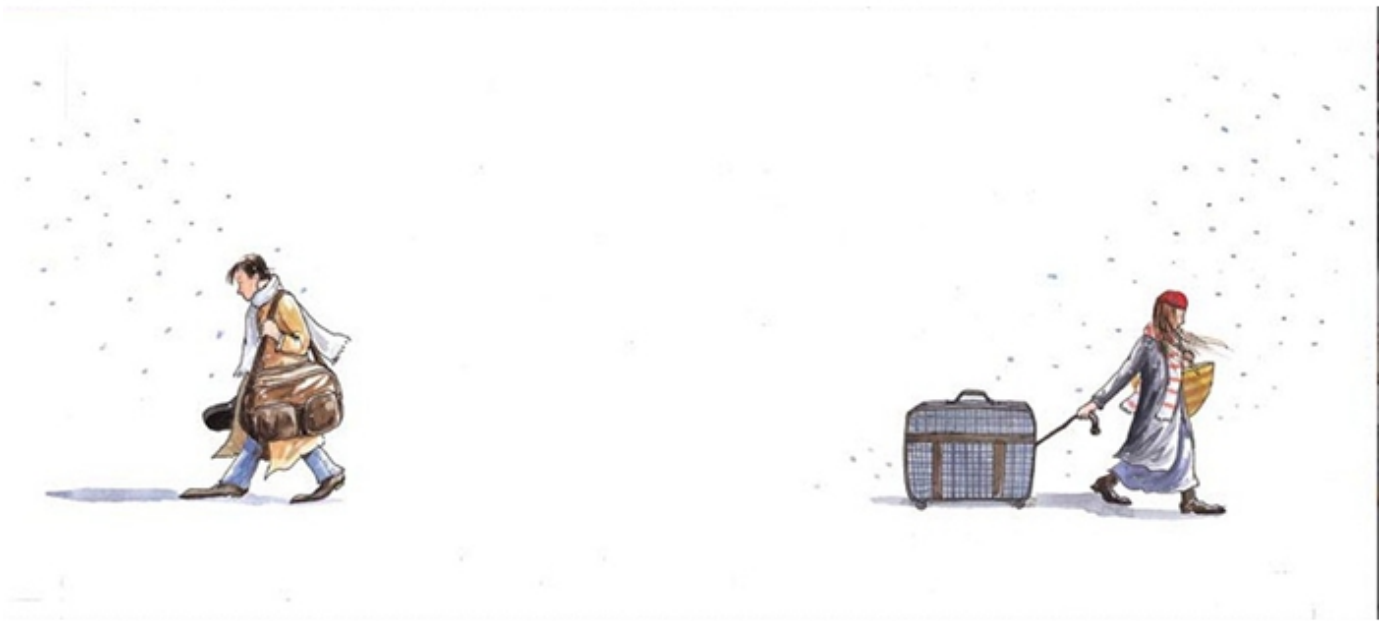
同时逻辑属性对于CSS盒模型也将带来相应的变化：



如果你在布局时，使用逻辑属性来替代物理属性，对于一些国际网站，比如阿拉伯语的网站，有着明显的作用。在改变书写模式时，不需要额外调整布局相关的属性：



这个我在《Web中向左向右》一文中做过详细的阐述。



有关于CSS盒模型更详细的介绍还可以阅读：

- [Web布局：CSS 盒模型](#)
- [图解CSS：CSS逻辑属性](#)
- [CSS的逻辑属性对盒模型带来的变化](#)

布局

Web布局对于Web前端开发者而言，它就是一个永恒的话题。

随着Web技术不断向前推进，Web布局相关技术也在Web不同的演化过程也有相应的演进。

在Web布局整个演进过程当中，经历了没有任何布局、表格布局、**定位布局**、**浮动布局**、**Flexbox布局**等布局模式。除了这些我们常看到的布局之外，即将还会有**Grid**、**Shapes**（类似杂志不规则布局）和**多列布局**（类似报纸排版布局）等现代布局模式。这些布局模式从侧面也反映出其自身是Web演进过程中的一种产物，都承载了自己在当时那个时期的使命。

对于Web前端开发人员而言面对Web的布局始终跟着网页的设计在走。而网页的设计在不同的时期也在不断的变化：从最初的站点到现在流行的站点在设计的发展有三个阶段：

Web Design 1.0 — 1996



The WorldWideWeb (W3) is a wide-area [hypertext](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), [November's W3 news](#), [Frequently Asked Questions](#).

What's out there?
 Pointers to the world's online information, [subjects](#), [W3 services](#), etc.

Help
 on the browser you are using

Software Products
 A list of W3 project components and their current state, (e.g. [Line Mode](#), [XSL](#), [NEXSTEP](#), [Servers](#), [Tools](#), [Mail robot](#), [Library](#))

Technical
 Details of protocols, formats, program internals etc

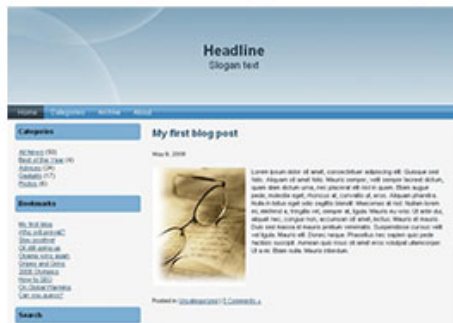
Bibliography
 Paper documentation on W3 and references.

Tables & Borders

2000



Blog Design — 2008

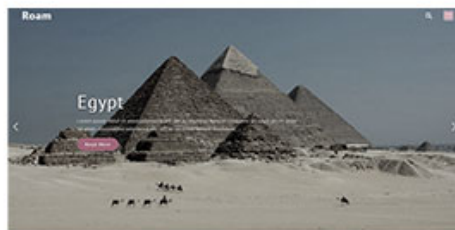


2010

Web Design 2.0

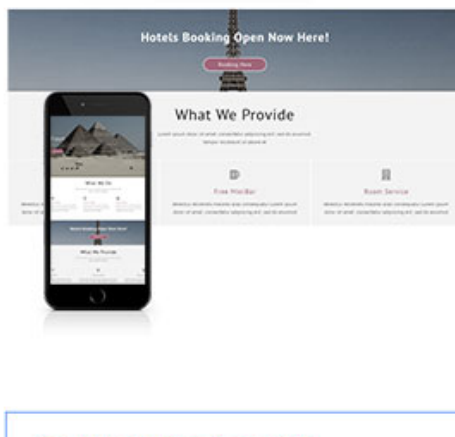


Bootstrap Design — 2015

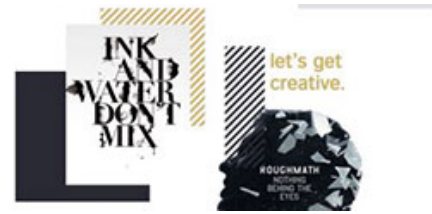


2019

Web Design 3.0

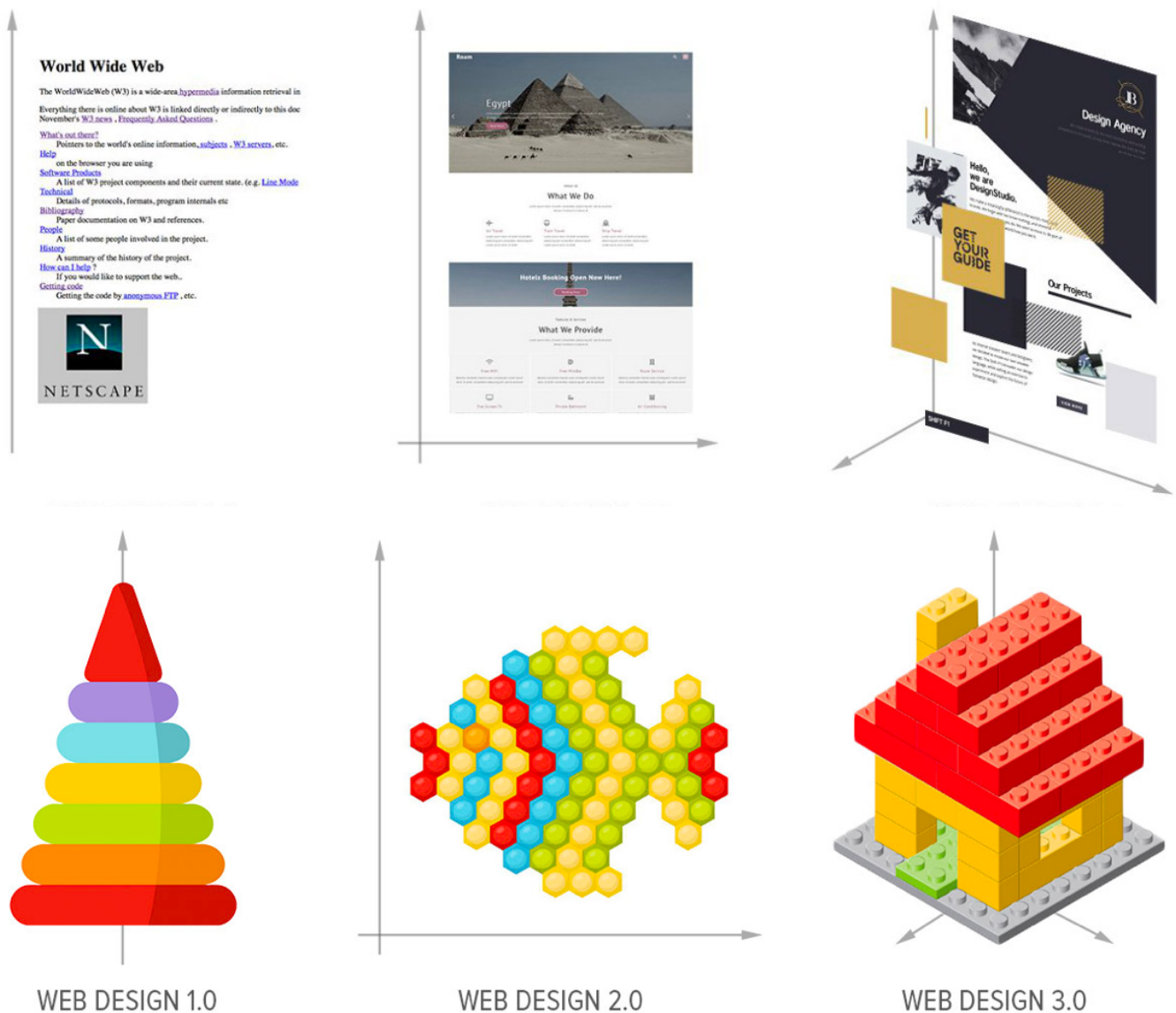


- Web Design 1.0
- Web Design 2.0
- Web Design 3.0



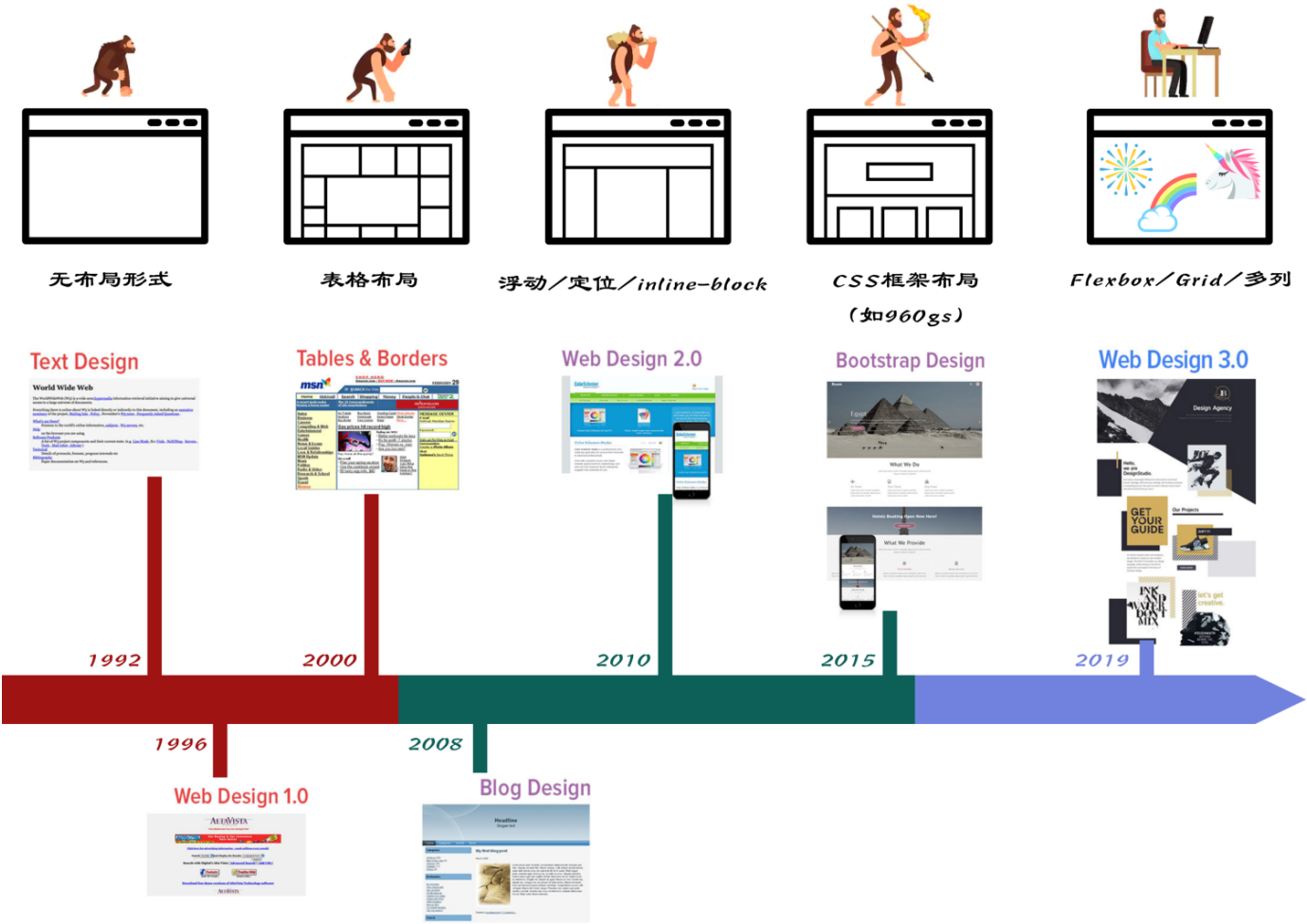
不同是代的设计情景之下，Web的元素定位也有其演变过程：

- Web设计1.0是“一维的”：设计元素大多是按顺序排列的（按文档流的自然顺序排列）
- Web设计2.0是“二维的”：单元格中有放置元素的网格，具有更多的自由性
- Web设计3.0是一个“新的维度”：它可以像平面设计工具一样的自由地定位元素、重叠。为网页设计开辟了新的前景。也将开启新的Web页面设计时代

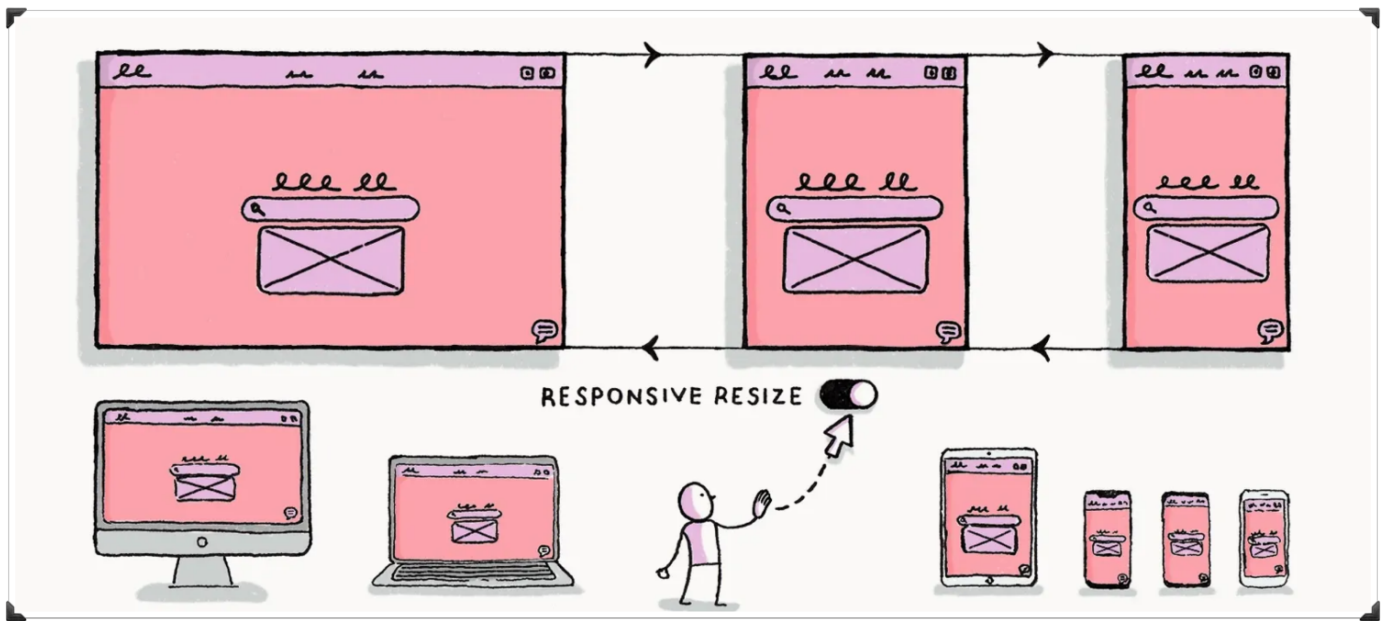


如果我们用乐高积木来形容的话，会更为形象一些，如上图所示。

针对不同阶段，Web布局相对应的模式也可以套入整个设计演变过程，如果有人猿进化来描述的话，有点类似下图：



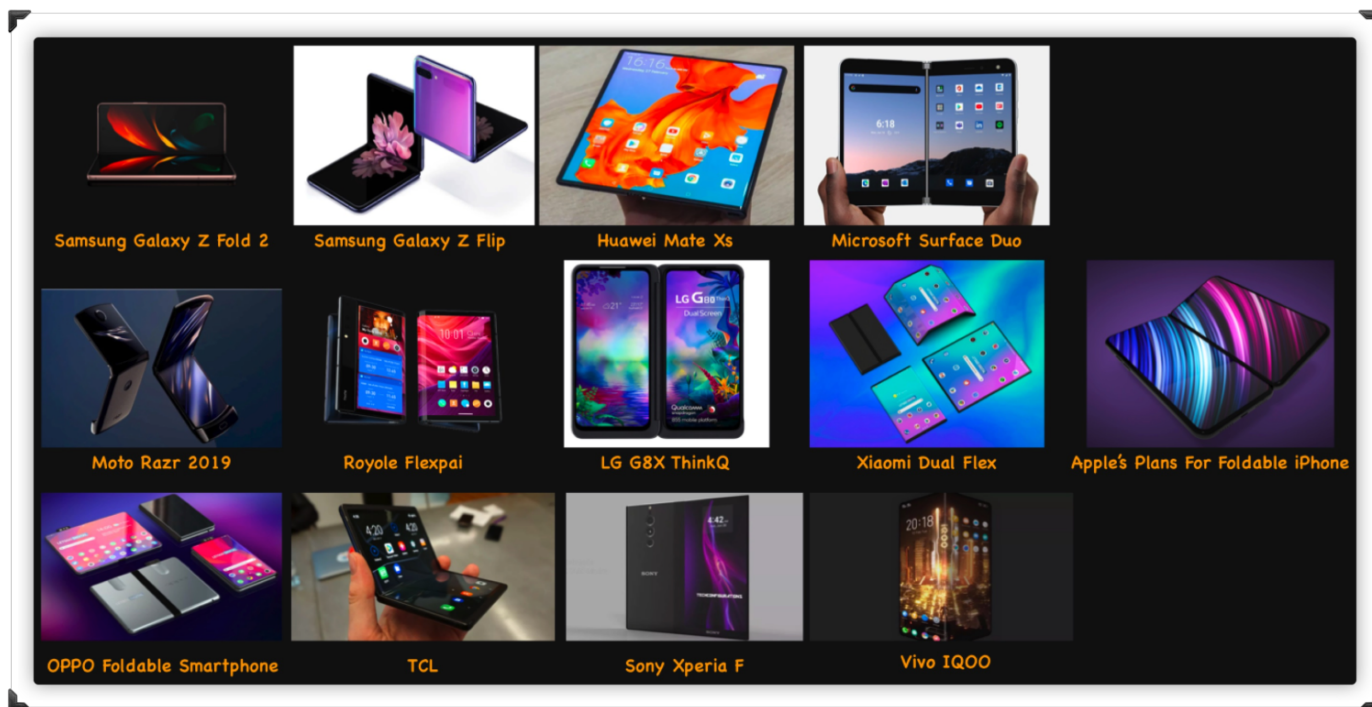
时至今日，已是移动端的天下，我们需要针对的设备类型也多起来了，为了满足更多的设备终端，我们还可以使用[响应式Web布局](#)：



在社区中只针对移动端的适配方案的讨论也非常地多，比如Flexible布局，VW布局等，如果你单独对移动端适配感兴趣的话，还可以阅读下面几篇文章：

- [使用Flexible实现手淘H5页面的终端适配](#)
- [再聊移动端页面的适配](#)
- [iPhone X的缺口和CSS](#)
- [iPhone X的Web设计](#)
- [如何在Vue项目中使用vw实现移动端适配](#)
- [移动端上的设计和适配](#)
- [手淘Web页面Bar和纵向适配的设计](#)

但这并没有终止Web布局的讨论。或许你不久就需要面对折叠设备或多屏设备的布局适配：

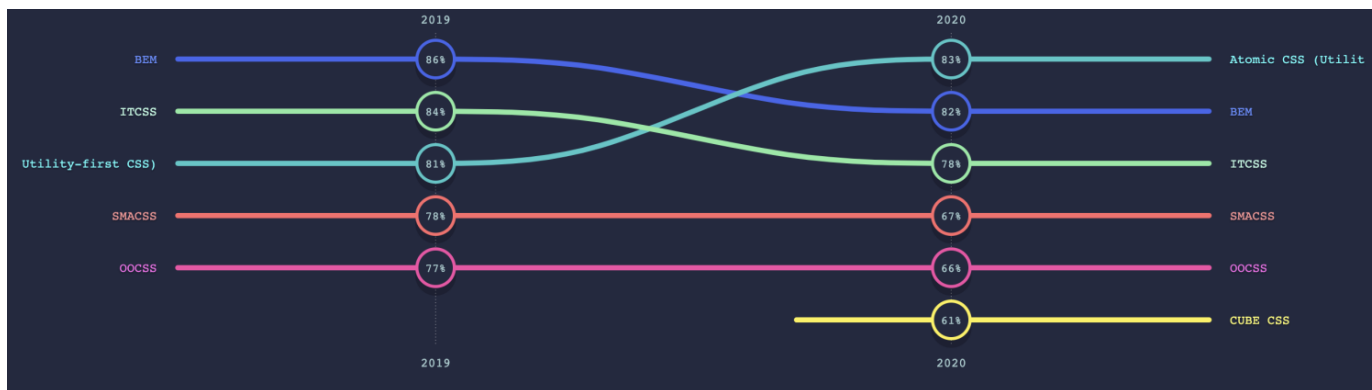


多屏和折叠屏设备的出现，有可能是下十年Web开发方式，设计和交互的模式都将带来改变。同样的，有关于这方面设备的CSS相关规范也开始在草案阶段，如果你对这方面感兴趣的话，可以阅读下面这几篇有关于折叠屏幕和多屏幕相关的技术文章：

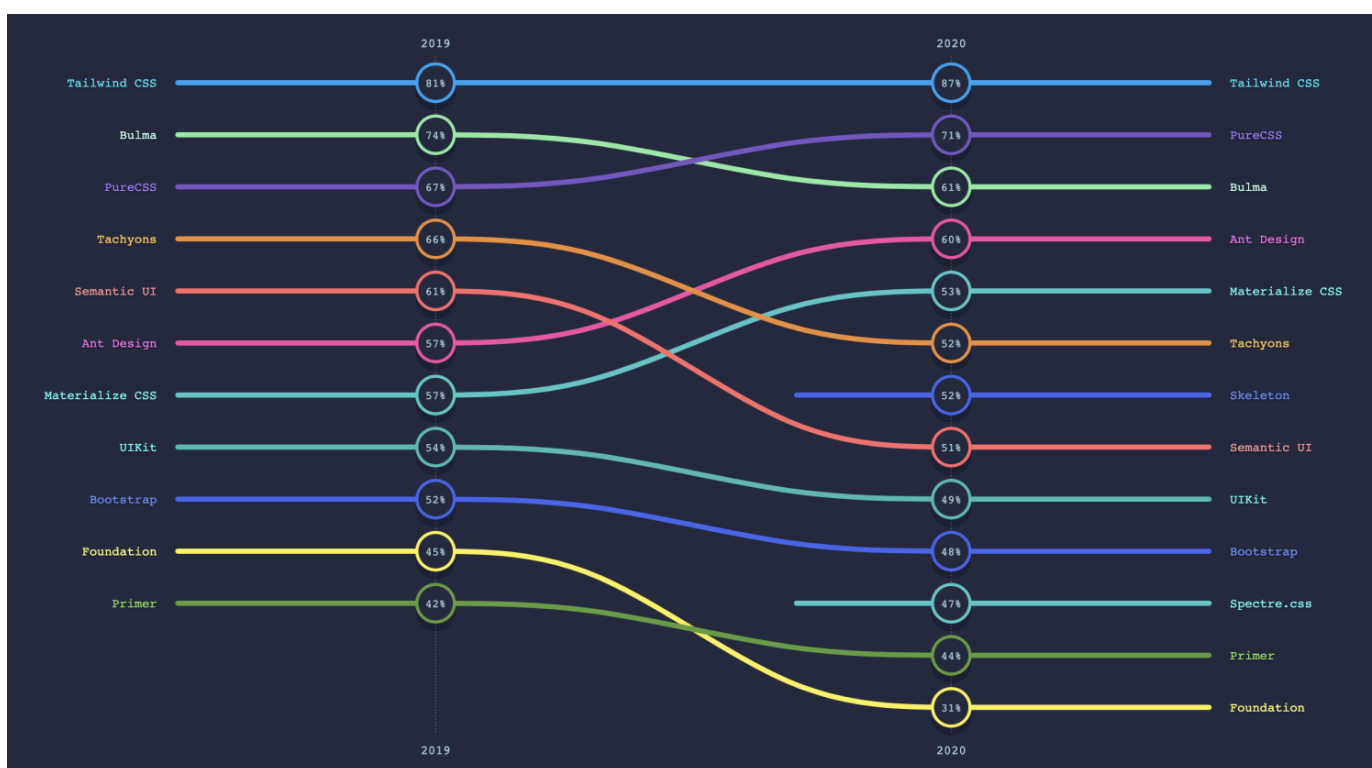
- [聊聊安卓折叠屏给交互设计和开发带来的变化](#)
- [可折叠Web可能会给我们带来的变化](#)
- [可用于双屏幕和折叠屏的Web API](#)
- [折叠屏相关的Web API](#)

写CSS的姿势

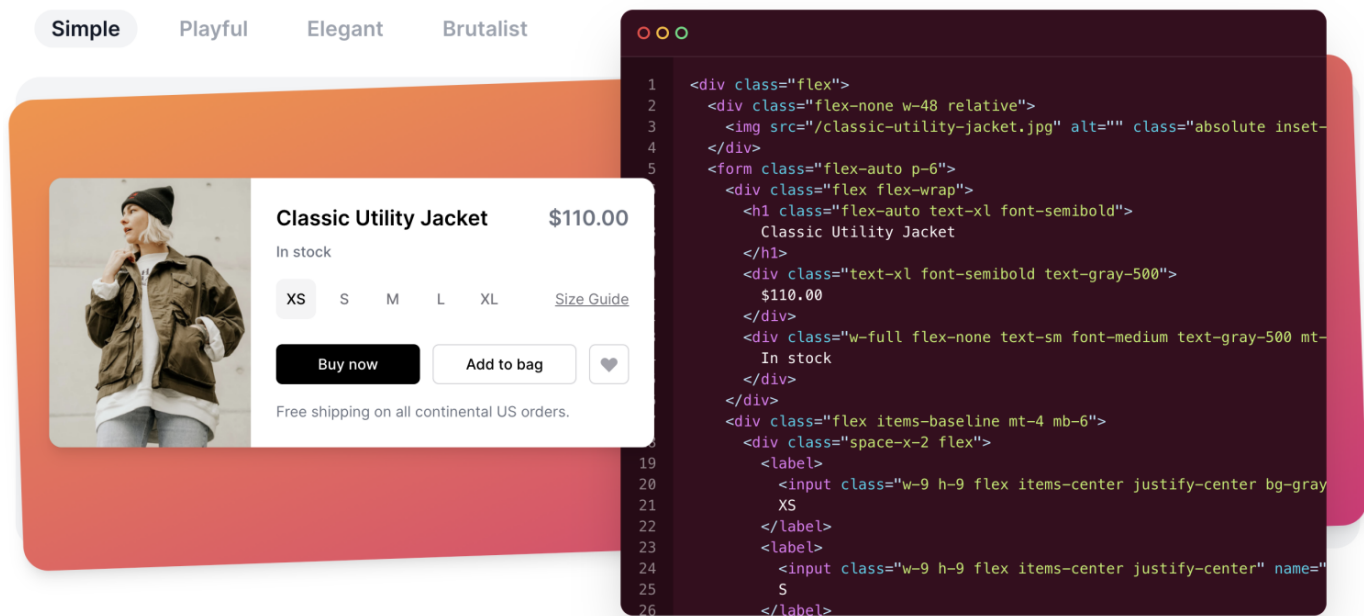
正因为CSS的层叠、权重、继承等因素，很多开发者在编写CSS的时候总是会碰到选择器冲突、样式覆盖、代码冗余等种种问题。为此，社区中有很多关于书写或者说[维护CSS的方法论](#)，比如：



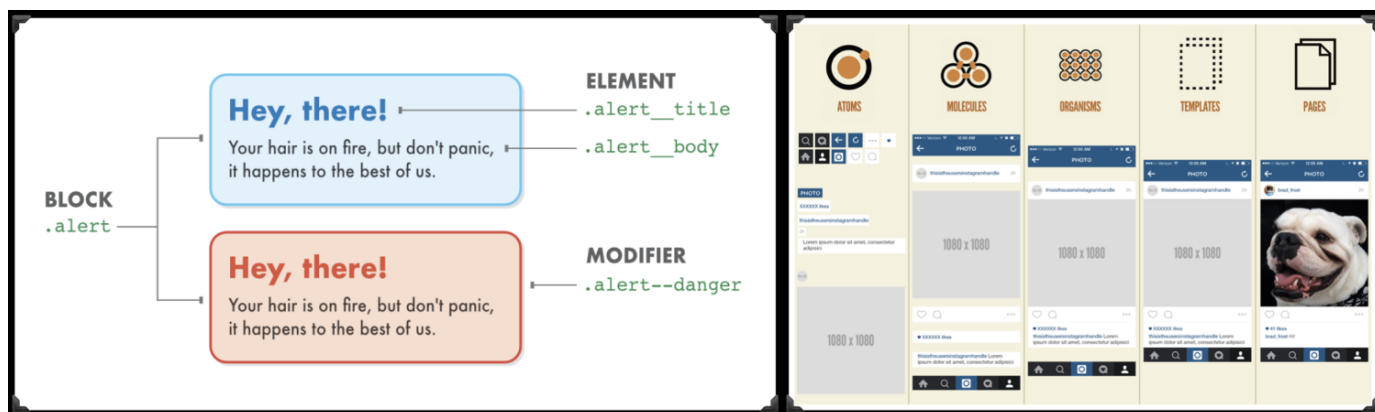
也有很多相关的CSS框架（CSS Frameworks）来帮助大家快速编写CSS，构建项目：



特别是这两年，社区讨论比较多的是Utility-first CSS，而这方面的代表框架就是Tailwind CSS：

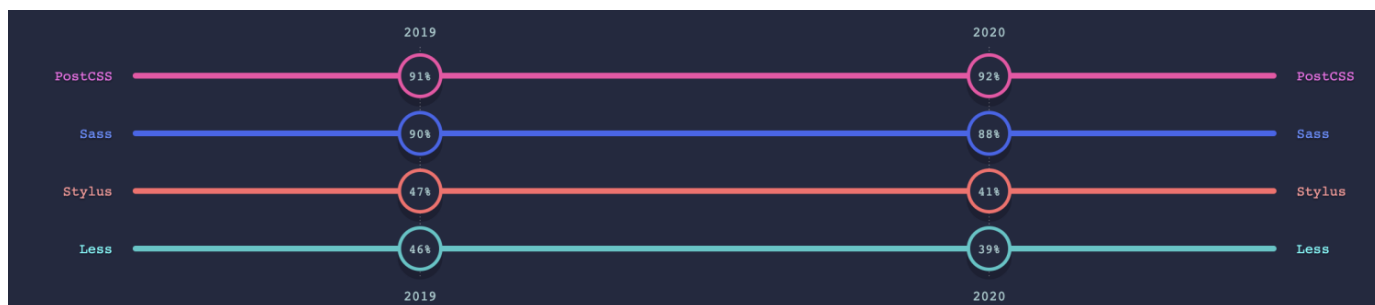


就个人而言，我比较喜欢BEM和Atomic：



现在我喜欢的是Utility-first CSS是Tailwind CSS。

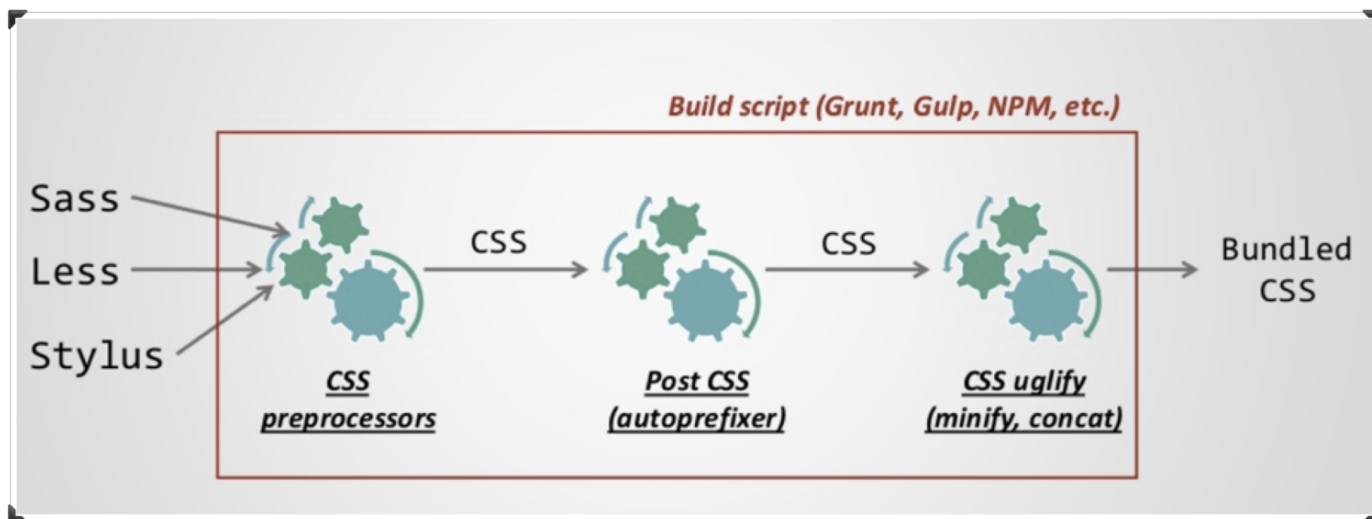
除了CSS的方法论和框架，为了提高大家的编码效率和维护项目的CSS，社区中还有很多CSS方面的处理器：



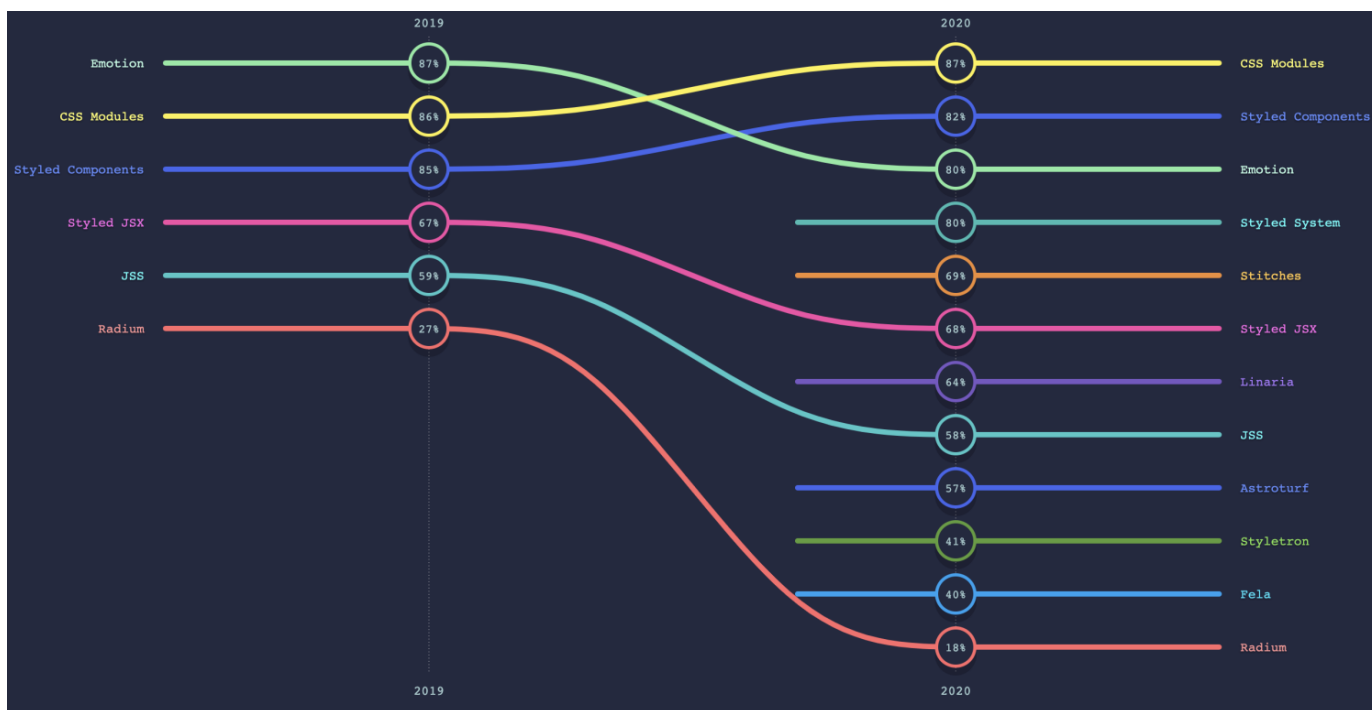
很多时候，Sass、Stylus和LESS常被称为预处理器，PostCSS常被称为后处理器，他们可以说是一前一后：



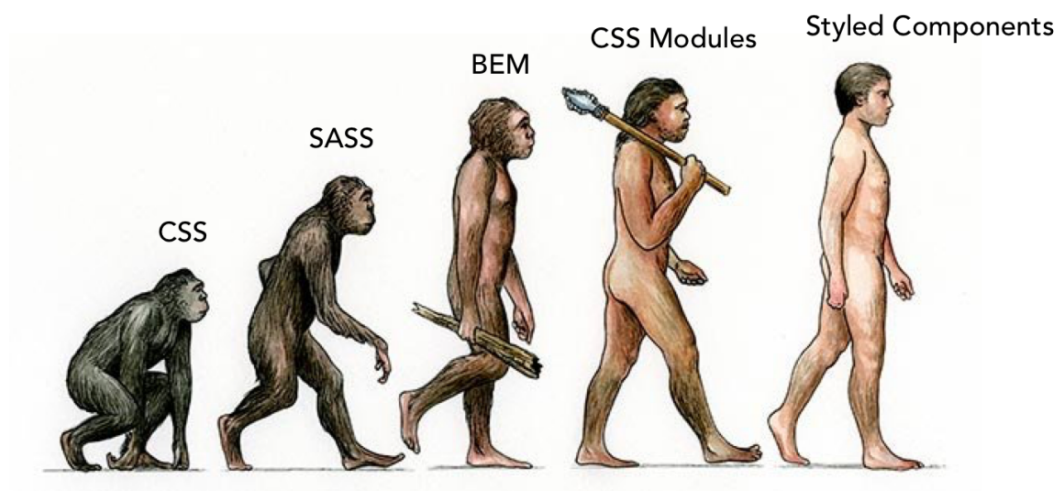
他们之间的关系如下图所示：



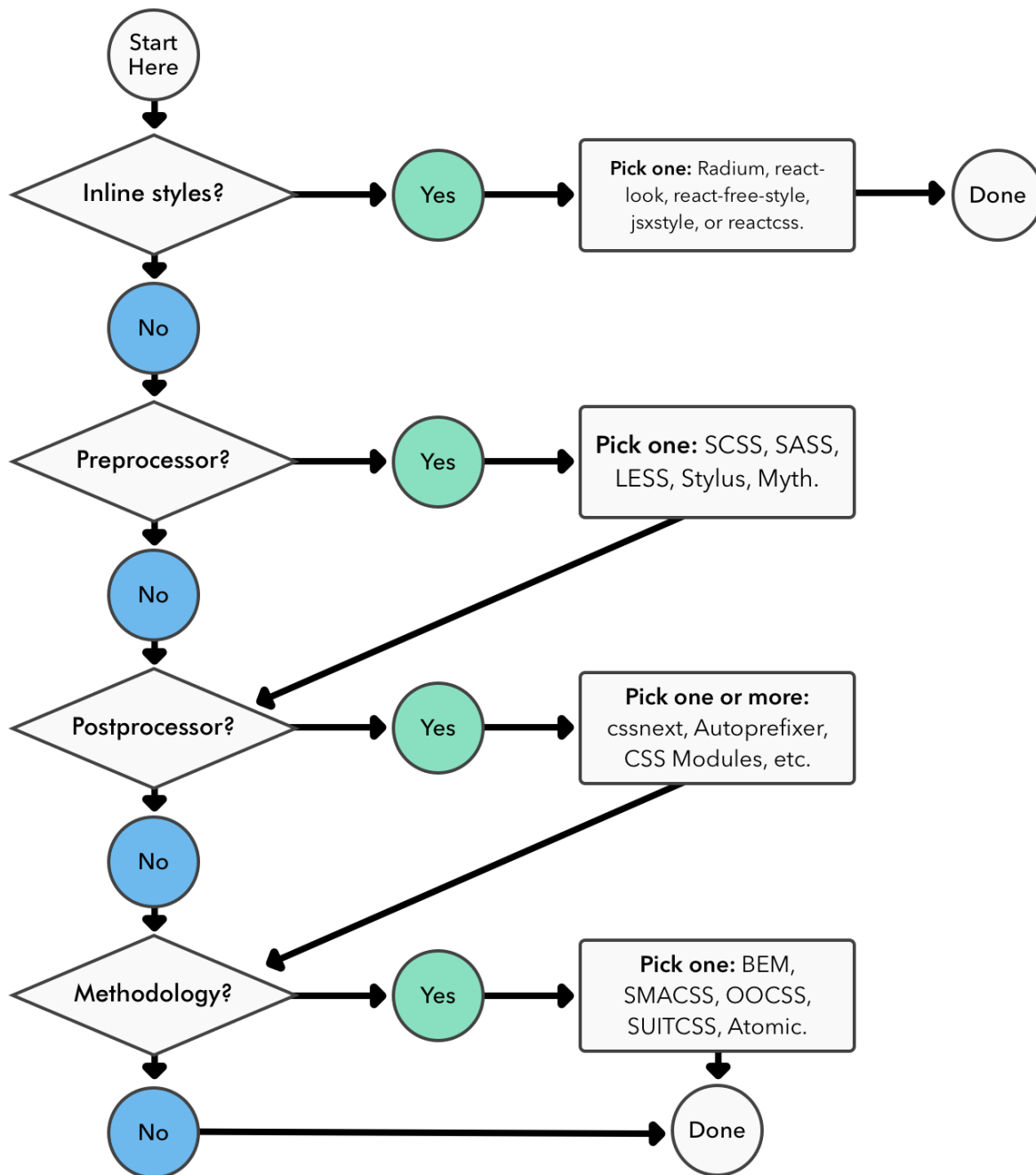
不过，随着React和Vue这样的框架出来之后，社区中出现另一种声音，那就是在JavaScript中编写CSS（即：CSS-in-JS）：



经历的这些种种，CSS的编写和维护也发生了很大的变化：



不过，不管怎么变，我们在编写CSS时，都可以像《[写CSS的姿势](#)》文章末尾提到的流程图，大家根据自己和自己的团队找到最为适合的方式。因为没有最好，只有最合适：



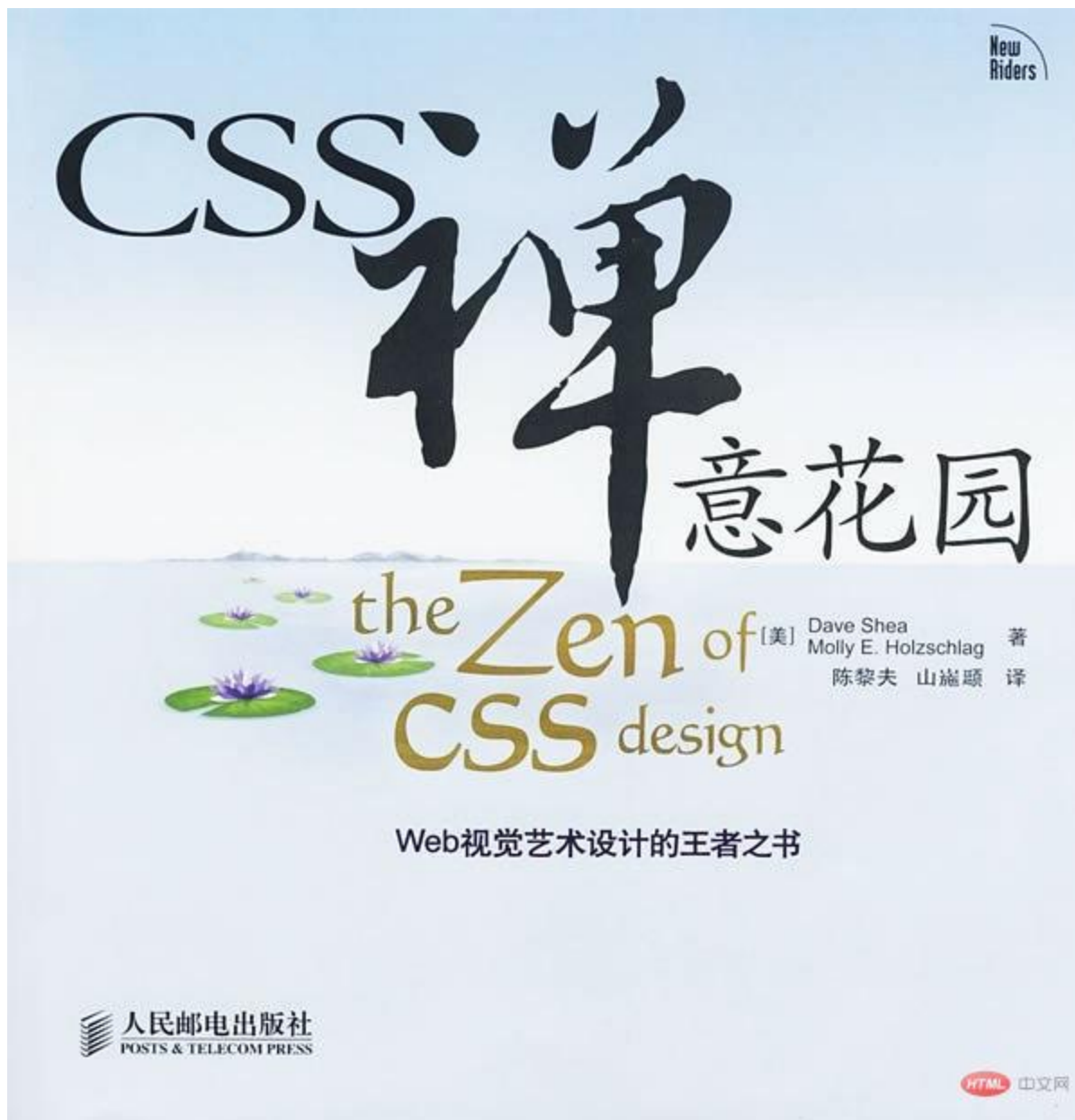
CSS Is Awesome



@Brandon Smith 有一篇博文是专门说 [CSS is Awesome!](#)

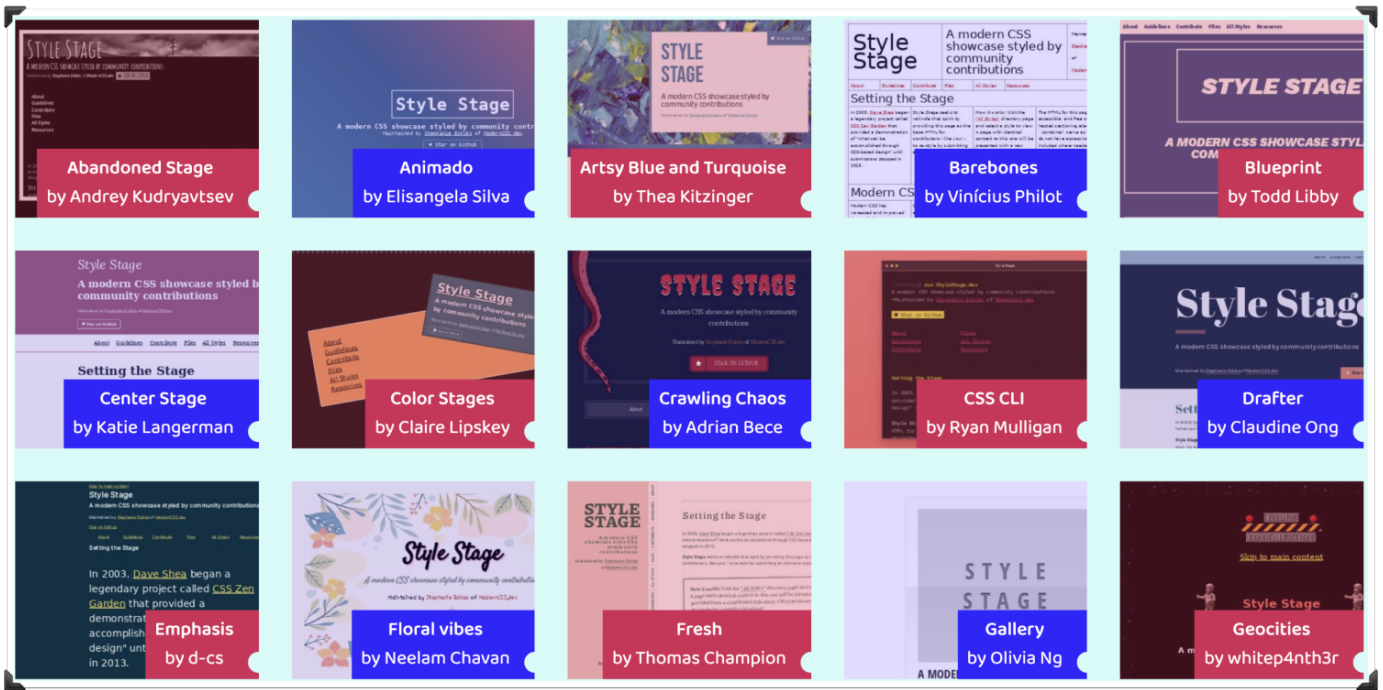
为什么说CSS非常棒（非常强大）呢？我从我自己的角度来和大家聊这个话题。

有一点年纪的CSSer（或者说Web开发者）对于[CSS禅意花园](#)应该不会感到陌生。CSS禅意花园有一个最大的特色，那就是基于同一套HTML的结构，可以实现不同的Web UI效果，即，有着不同的布局风格：



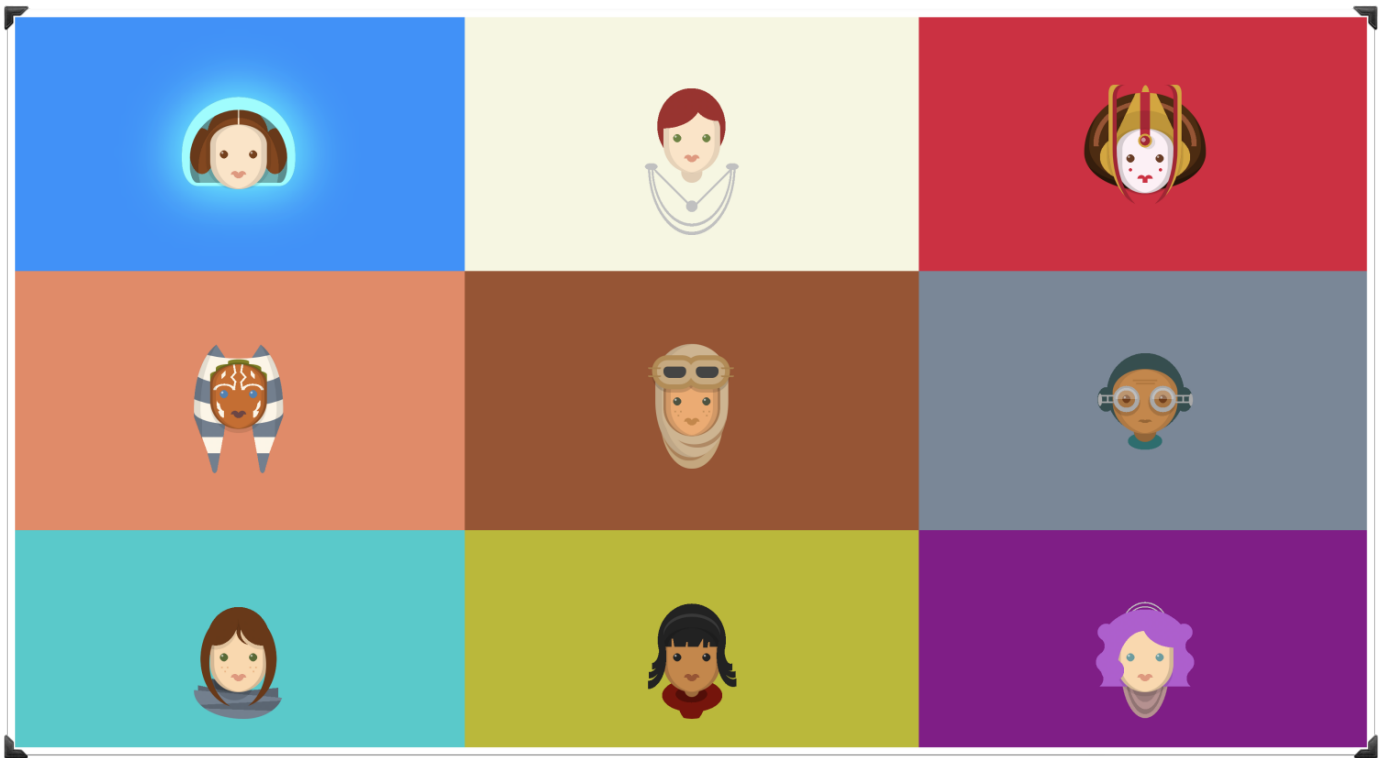
感兴趣的同学不妨也去试试，看看自己能基于同一套HTML结构，能实现多少种不同的UI效果。

另外一个和CSS禅意花园相似的就是 @Stephanie Eckles 推出的 [Style Stage](#)，也可以基于同一HTML结构，实现 同的UI效果：



我把她称为现代版的CSS禅意花园，现代版的CSS设计之美。

时至今日，在现代Web开发中，CSS越来越强大，她的作用不仅仅是实现Web布局，简单的UI效果。我们可以使用CSS实现项目中很多复杂的UI效果：



更不可思议的是，使用CSS还可以绘画：

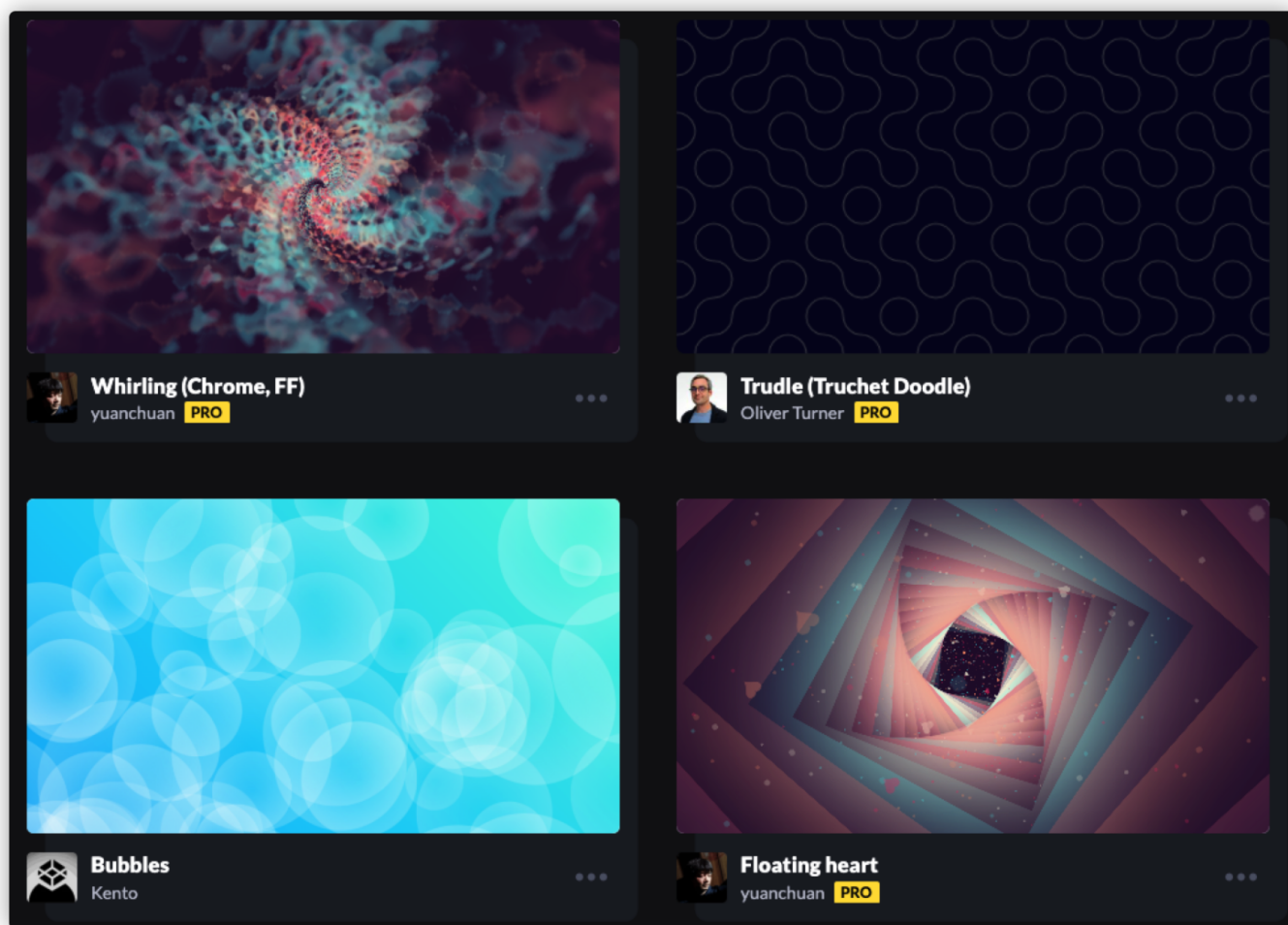


在我国内 @袁川 老师在这方面也很有研究，比如袁川老师在CSSConf（中国）分享的《[Generative Art With CSS](#)》的话题：

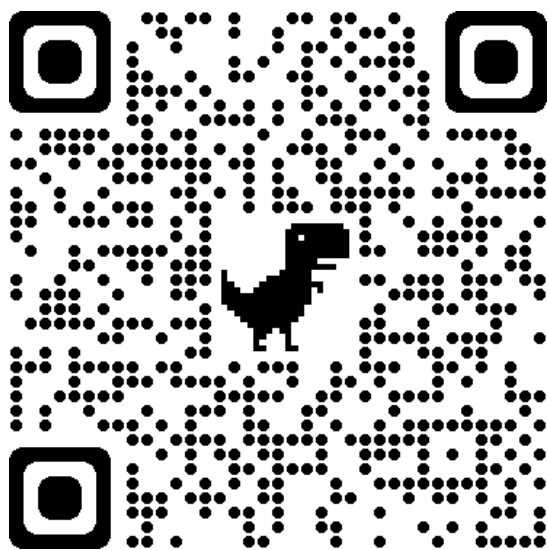


Generative Art With CSS

给大家展示了如何使用以 `<css-doodle>` 绘制不同的艺术作品，你会领略到CSS神奇的另一面，也会感叹到原来CSS离艺术的创作是这么的近：



除此之外，近几年CSS发展也特别的快，推出很多新特性。比如@argyleink在伦敦（LondonCSS 2020）CSS第四次活动中分享的一个话题《What's new with CSS?》，就提到了很多个CSS的新特性：

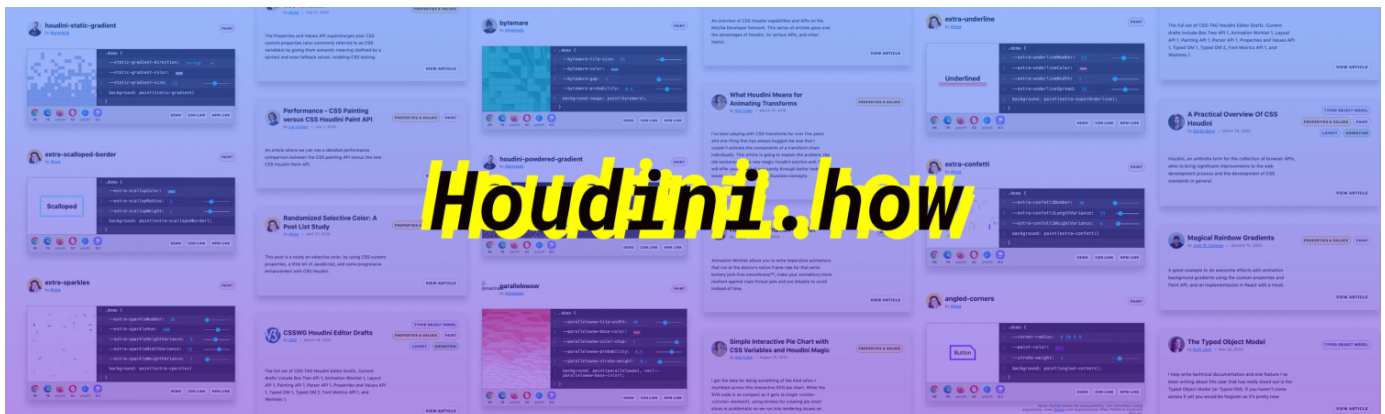


在PPT中提到：

- CSS自定义属性
- CSS逻辑属性
- CSS书写模式
- CSS的 `gap` 属性
- CSS的Grid布局, 以及 `subgrid`
- CSS的滚动捕捉
- CSS新的媒体特性
- CSS选择器Level 4

我在该基础上重新整理了一篇新的文章, 在文章中介绍了近24个有关于CSS方面的特性。

另外, 在分享的时候, 还遗漏了一个话题, 那就是**CSS的Houdini**, 我自己对这方面非常的感兴趣。今年的GDS大会上 @Una Kravets 就分享了一个这方面的话题, 而且还向社区推出了**houdini.how**网站, 收集了CSS Houdini中Paint API和自定义属性构建的Demo集:



在houdini.how提供的Demo, 我们都可以通过调整自定义属性值, 看到不同的效果:

**extra-confetti**by [@Una](#)

PAINT



```
1 .demo {  
2   --extra-confettiNumber: 225  
3   --extra-confettiLengthVariance: 12  
4   --extra-confettiWeightVariance: 4  
5   background: paint(extra-confetti)  
6 }
```

65 79 polyfill 52 polyfill 9.2

DEMO CDN LINK NPM LINK

**houdini-static-gradient**by [@argyleink](#)

PAINT



```
1 .demo {  
2   --static-gradient-direction: to-top  
3   --static-gradient-color: #ff00ff  
4   --static-gradient-size: 16px  
5   background: paint(static-gradient)  
6 }
```

65 79 polyfill 52 polyfill 9.2

DEMO CDN LINK NPM LINK

如果你对这方面感兴趣的话，[也可以把自己构建的Demo提交到Github的仓库中。](#)

如何学习CSS

接下来简单地聊一下，我自己是怎么学习CSS的，仅是自己的一点小心得，仅供参考。

我想不管是学习什么知识，应该都离不开书吧！就CSS方面，我觉得有几本书是很值得大家花点时间阅读的，比如：



如果你已不是初级的CSSer，那么W3C中有关于CSS相关的规范文档是值得一读：

W3C Cascading Style Sheets home page

HIGHLIGHTS

- CSS Writing Modes is a W3C Recommendation [press release]
- 1 October 2019: W3C is 25 years old.
- The "CSS Snapshot" (latest: 2018) lists the parts that are ready for implementers

WHAT IS CSS?

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents.

These pages contain information on how to learn and use CSS and on available software. They also contain news from the CSS working group.

STANDARDS & DRAFTS

Some of the specifications and drafts by the **CSS Working Group**:

Completed work	Status	Upcoming
CSS Snapshot 2018	NOTE	
CSS Snapshot 2017	NOTE	
CSS Snapshot 2015	NOTE	
CSS Snapshot 2010	NOTE	
CSS Snapshot 2007	NOTE	
CSS Color Level 3	REC	REC

WORKING GROUP NEWS

The working group regularly publishes reports on its blog; here are the most recent:

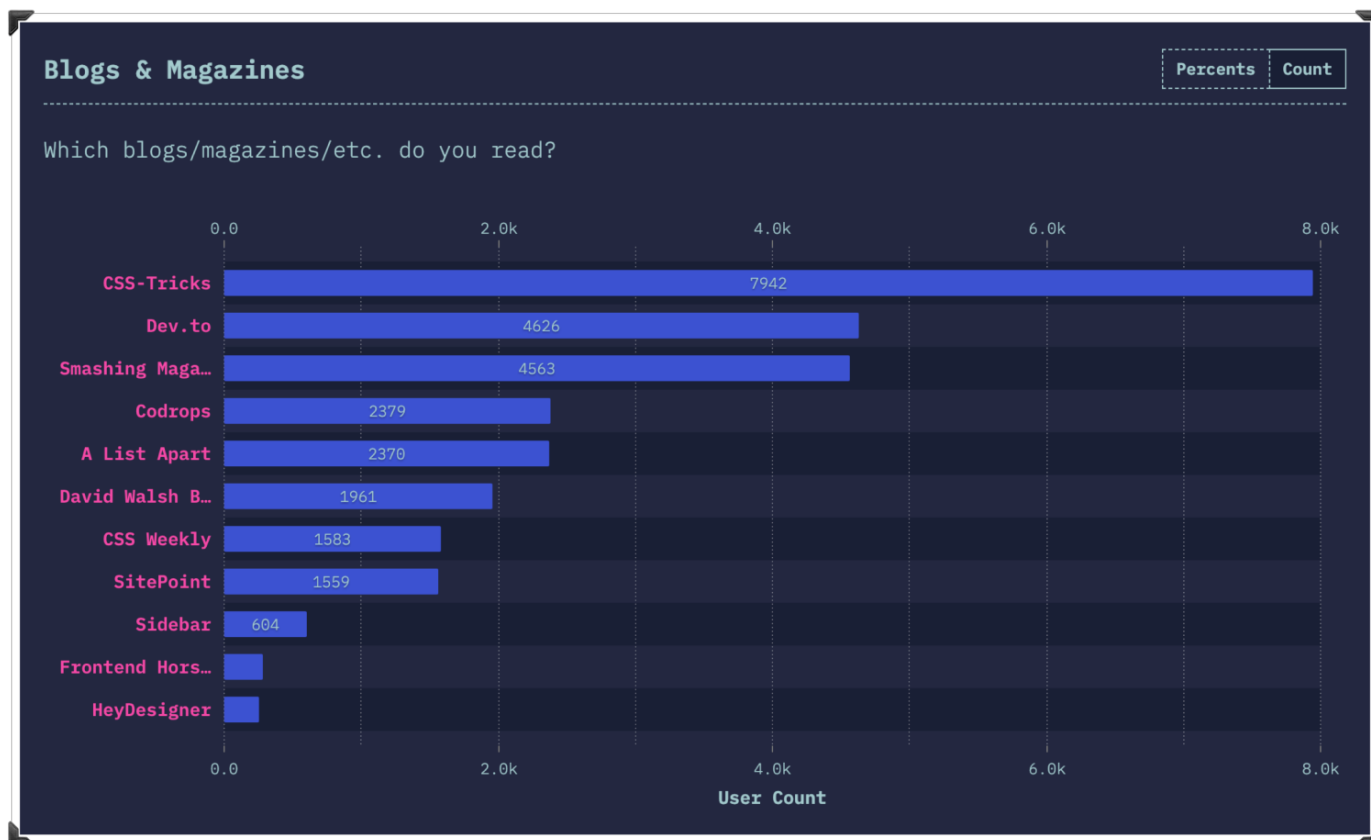
- Minutes Telecon 2020-12-09
- Minutes Telecon 2020-12-02
- Minutes Telecon 2020-11-18
- Updated CSS Lists and Counters Level 3
- Minutes Telecon 2020-11-11
- New Working Draft of CSS Values and Units 4 Published
- Minutes Telecon 2020-11-04
- Minutes Telecon 2020-10-07

NEWS

11 DEC 2020 RealObjects released PDFReactor version 11, an XML-to-PDF formatter that runs either as a Web service or as a command line tool. It is also available as a Java library. It has support for; among other things, CSS Transforms, CSS Regions, Web Fonts, and running elements. Other features include support for HTML5, MathML, SVG, XSLT, JavaScript, PDF/A and accessible PDF. Version 11 adds, among other things, support for CSS grid, page floats, custom counter styles and OpenType ligatures. (Java. Free demo version.)

说实话，阅读规范是件痛苦的事情，但不同的时期，不同的阶段去阅读规范都会有不同的收获。好比我自己，我今年重新阅读这些规范时，收获就不少。可能阅读规范更多关注点是CSS属性的使用，但近一年来重新阅读规范时，我更关注的是属性使用的临界点相关的知识。换句话说，我们在平时使用CSS时碰到的问题，其实在规范中都有相应的描述，也能找到相应的答案。

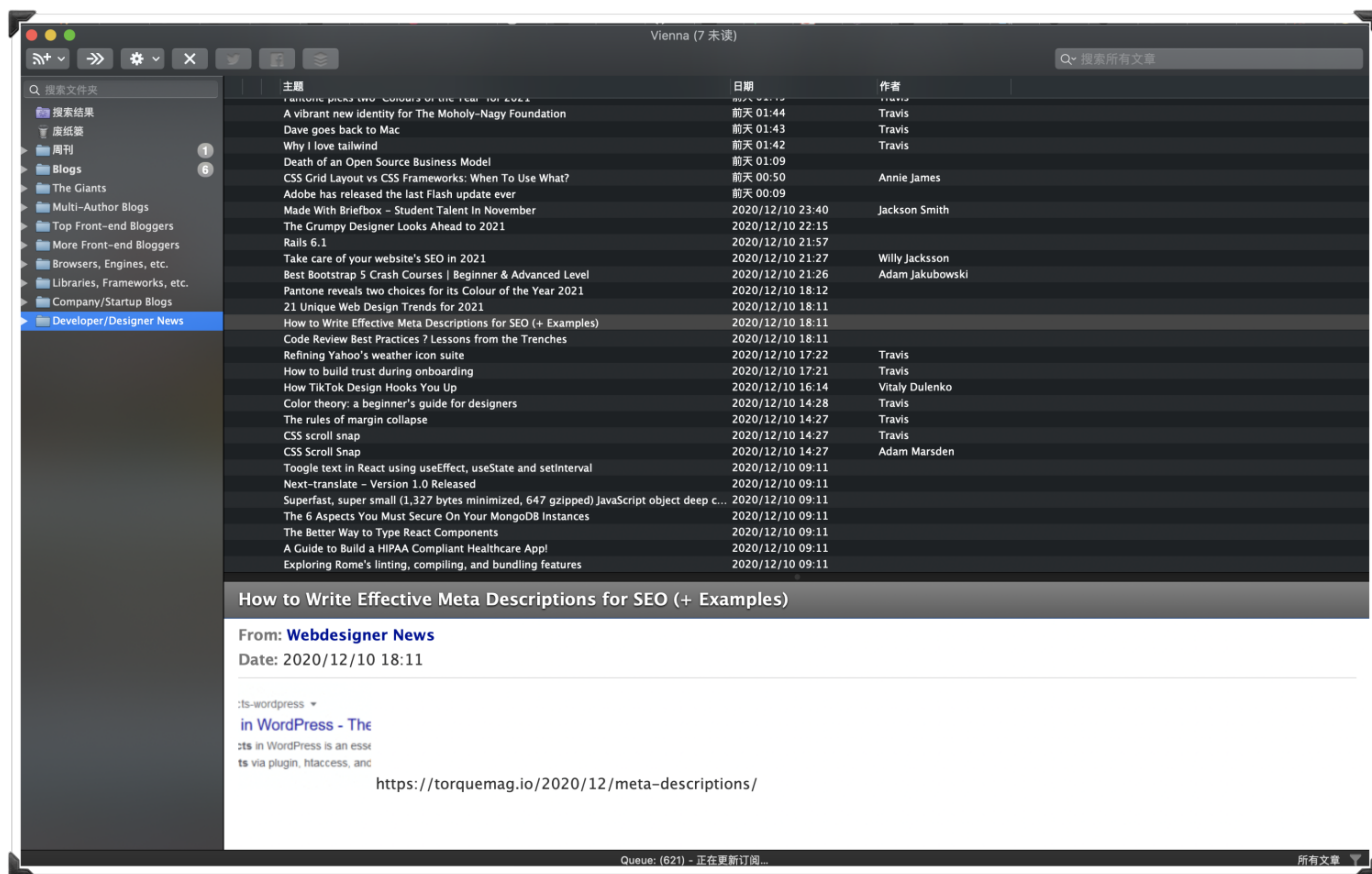
除了阅读规范之外，社区中很多优秀的博客也是值得我们去阅读：



在中国社区，有关于CSS方面的博客除了我自己的[W3cplus](#)之外，特别推荐：

- 张鑫旭 老师的 [鑫空间](#)，[鑫生活](#)
- chokcoco 老师的 [爱CSS \(ICSS\)](#)

在社区中发现好的博客或者站点时，你还可以使用RSS应用来订阅它们：



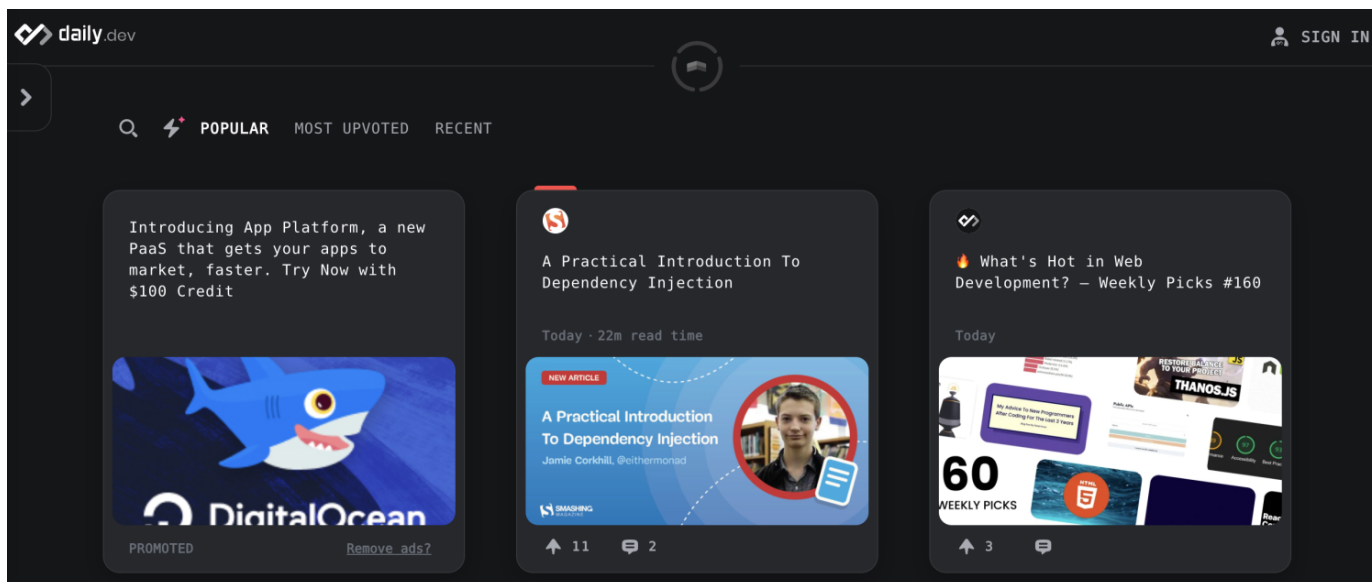
RSS是一个好东西，她能帮你省下不少的时间，而且获取信息更有针对性，都有可能是你自己喜欢的内容。

附件是我自己收集的一个RSS集合：

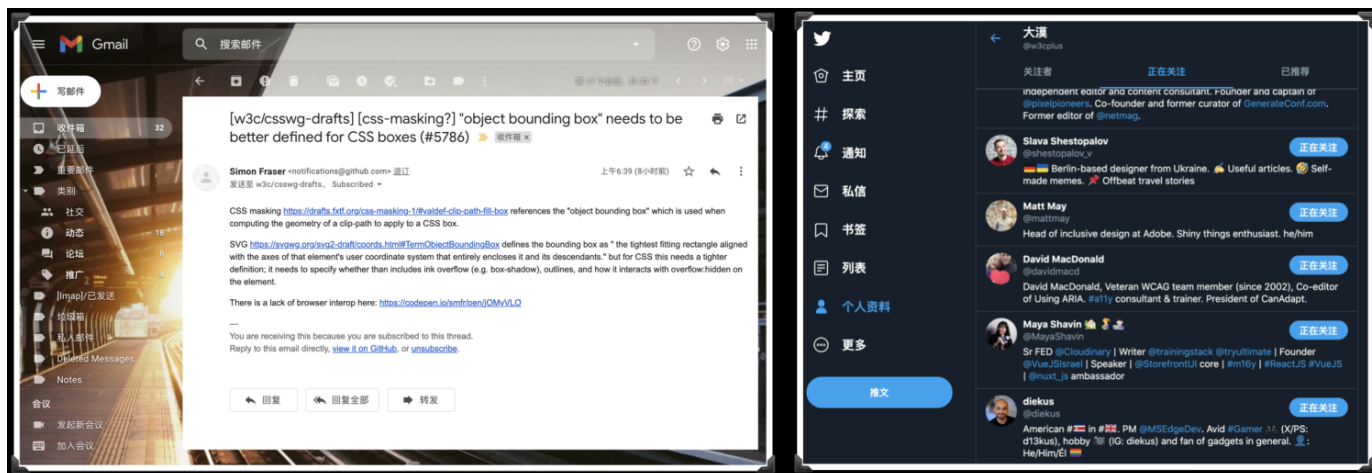
 [rss.opml.pdf](#)

下载下来，请把 `.pdf` 后缀删除，在RSS应用中导入 `rss.opml` 文件即可。你也可以此基础上新增或删除。

如果你不太喜欢使用RSS的话，也可以使用一些浏览器的扩展插件，比如Chrome浏览器，我就整了一个 [Daily.dev](#)：



获取信息还有另一种方式，那就是订阅Github上感兴趣仓库或者关注行业内的一些大神：



获取信息渠道很多，造成获取信息成本很大。很多时候可能今天看到了好的话题，只是草草瞄了一下，想在有时间的时候深读，却在这个时候找不到文章的链接了。我就常犯这样的毛病，为此我这两年养成了一个习惯，在本地创建了一个本地Blog，其中有一个栏目就是收集每个月自己认为有意思的文章：

2020年资源集

2020-01
2020-02
2020-03
2020-04
2020-05
2020-06
2020-07
2020-08
2020-09
2020-10
2020-11
2020-12

首页CSSA11YJavaScriptReactSVGPL性能动效SlidesWeb技巧Demos资源

2020年资源集

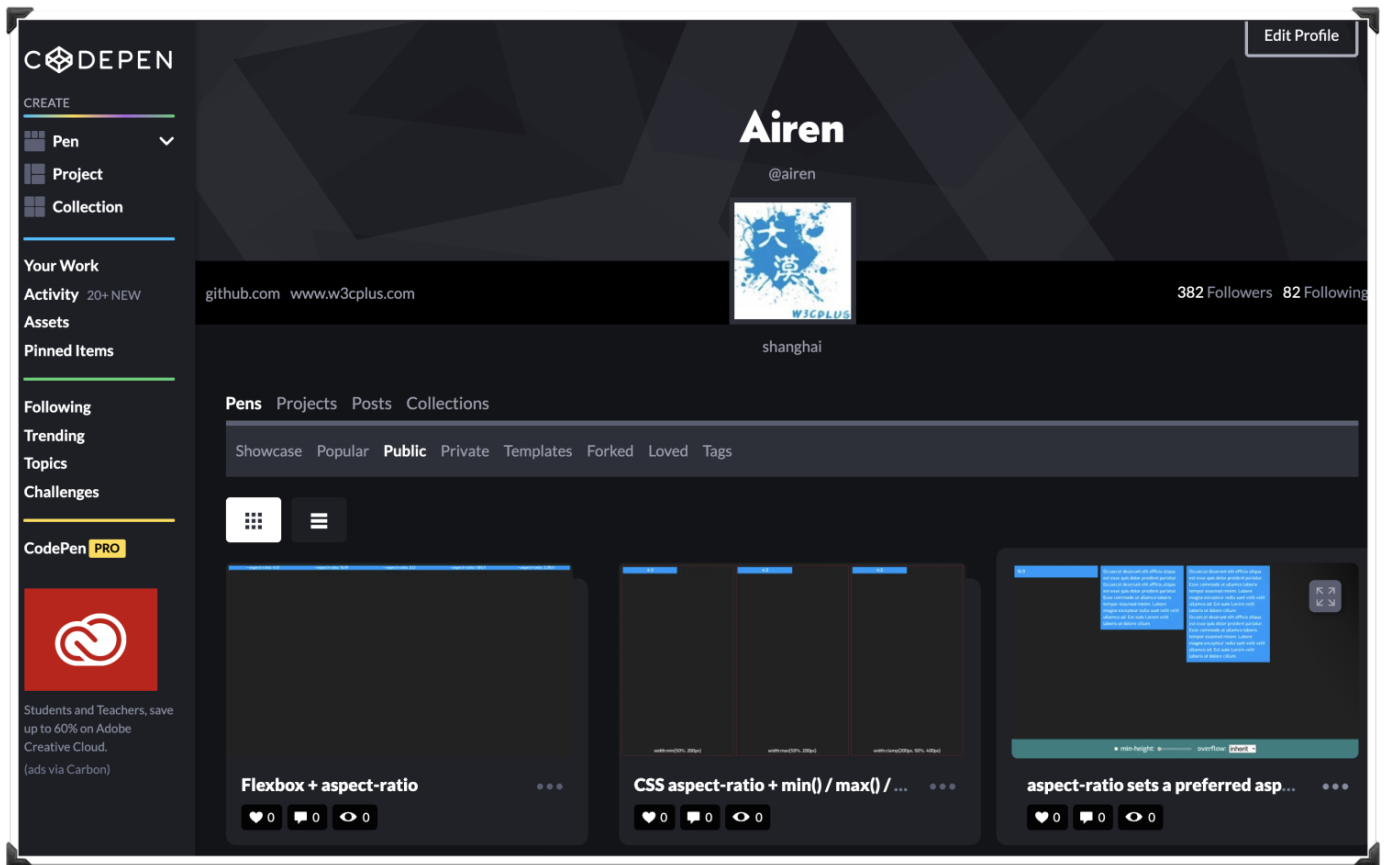
这里将会涵盖Web相关的系列资源，为了更好的区分资源类型，将以不同的方式来区分：
JavaScript(🐼)**、**CSS(W)**、**HTML(🍷)**、**TypeScript(🐍)**、**Vue(🔥)**、**React(🔥)**、
CSS Houdini(🍷)、**Animation(🎨)**、**Canvas(🎨)**、SVG(🍷)、**Web Component(🍷)**、
WebSockets(🍷)、**Performance(🍷)**、**Accessibility(🍷)**、**UI/UX(🍷)**和 WebGL(🍷)。已
阅读使用🍷✓，重要使用🔥，待阅读使用🍷-作标记！

2020-01

- [Beginner's Guide to SVG Morph](#) [🍷]
- 🍷✓[Accessibility: Skip to main content](#) [🍷]-
- [JavaScript Questions](#) [🍷]
- [The cost of JavaScript in 2019](#) [🍷]
- [Understanding breakpoints — How to deliver meaningful specs to developers](#) [W]
- 🍷✓[Understanding CSS Grid: Creating A Grid Container](#) [W]-
- 🍷✓[Get Moving \(or not\) with CSS Motion Path](#) [W]-
- 🍷✓[WCAG 2.1 Guidelines Explained with Examples](#) [🍷]-
- [Why You Should Choose HTML5 <article> Over <section>](#) [W]
- [Multi-Thumb Sliders: Particular Two-Thumb Case](#) [W]
- 🍷✓[Using easing for more than just CSS transitions](#) [W]-
- 🍷✓[Understanding CSS Grid: Grid Lines](#) [W]-
- 🍷✓[HTML attributes to improve your users' two factor authentication experience](#) [🍷] 🍷
- [Navigating the web with a gamepad](#) [🍷]
- [Adding separators to layouts with CSS-in-JS](#) [🍷]

上面说的都是看和读，仅此是不够的，我们应该还需要多写。

写Demo，我自己常常会在Codepen上写一些Demo，不管是验证性的，还是创作性的，或者说是练习性的。我都喜欢在上面写：



Codepen上除了可以让自己做练习之外，还可以看到很多Demo，从别人写的Demo中学到一些新知识，新技巧。如果你还没有开始体验的话，那么这里强烈建议你开启在Codepen之旅。

除了写Demo还是不够的，我们还应该针对自己学的知识点做总结，其中写博客就是很好的一种方式：

CSS专栏	☐ CSS布局实战:多列布局
2019-05	☒ CSS布局实战:Flexbox布局
2019-06	☐ CSS布局实战:Grid布局
2019-07	☐ CSS布局实战:不规则布局
2019-08	☐ CSS布局实战:响应式布局
2019-09	☐ CSS布局实战:其他布局
2019-10	☐ CSS布局实战:移动端布局
2019-11	
2019-12	
2020-01	
2020-02	
2020-03	
2020-04	
2020-05	
2020-06	
2020-07	
2020-08	
2020-09	
2020-10	
2020-11	
2020-12	
Web布局大全专栏	
图解CSS系列	
	图解CSS系列
	☒ 图解CSS: 揭开CSS的面纱
	☒ 图解CSS: CSS层叠和继承
	☒ 图解CSS: CSS的盒模型
	☒ 图解CSS: CSS的值和单位
	☒ 图解CSS: 元素尺寸的设置
	☒ 图解CSS: 条件CSS
	☒ 图解CSS: CSS自定义属性
	☒ 图解CSS: CSS逻辑属性
	☒ 图解CSS: CSS颜色
	☒ 图解CSS: 变量字体
	☒ 图解CSS: CSS中的函数
	☒ 图解CSS: Flexbox布局
	☒ 图解CSS: border-radius
	☒ 图解CSS: CSS渐变
	☒ 图解CSS: CSS媒体查询
	☐ 图解CSS: CSS滚动捕捉
	☐ 图解CSS: Grid布局
	☐ 图解CSS: 不规则布局
	☐ 图解CSS: 多列布局
	☐ 图解CSS: 书写模式
	☐ 图解CSS: CSS选择器
	☐ 图解CSS: CSS边框
	☐ 图解CSS: CSS背景

我自己一般会先对问题分类：

- 我懂的
- 我似乎懂的
- 我不懂的
- 我感兴趣的

针对这些都列出一些清单，那么就可以有计划的去查看一些东西，学一些东西，写一些东西。

到现在为止，应该是2020年最后一天了，回忆了一下，我自2010年创建W3cplus到现在，我坚持写了十年的博客，差不写了1561篇文章：



当然，你可能会觉得有些是无用的，但就我自己而言都是有用的。除了有写博客的习惯之外，[我有时候还会对自己的一年，或一个阶段做一些总结。](#)

每个人和每个人都会有所不同的，都可能有所自己的一套方法，我的学习方法可能比较笨，简单地说就是 **多看、多写、多问、多总结** 等。

AI会取替CSSer?

临近年底，我自己从呆了五年之外的互动团队换到了 [淘系F\(x\) Team 团队](#)。开启了新的征程，在新团队主要做的是一款技术产品：[imgcook](#)



在Github上也能找到哟！

imgcook是干嘛的，我这里不做过多的阐述，直接给大家上一个视频：

[屏幕录制 2020-11-12 下午5.03.13.mov](#)

如果你感兴趣的话，可以直接来先体验一波！如果你使用imgcook碰到任何问题，都可以使用钉钉扫下面的二维码：



最后留一个话题，希望对该话题感兴趣的同学可以一起来探讨：

“ 在不久的将来，AI会取替CSSer吗？ ”

这主是我这次和大家一起探讨的话题，谢谢大家阅读！

谢谢



这次分享的PPT：