

CSS 入门:基础知识梳理

👤: 润华

👤: 一帆

1. 基础概念

- 什么是 CSS

CSS 即层叠样式表 (Cascading Style Sheets)



(维基百科 [层叠样式表](#), 图片作者: [Fenring](#))

HTML 用于定义内容的结构和语义, CSS 用于设计风格和布局。

比如, 您可以使用 CSS 来更改内容的字体、颜色、大小、间距, 将内容分为多列, 或者添加动画及其他的装饰效果。([MDN](#))

- CSS 主要目标

除了对网页进行装饰，`CSS` 还有一个主要目标就是 **将内容与显示分离**。

CSS 出现之前，如果需要为 `HTML` 元素添加样式，那么往往需要用到其他的样式元素如 ``，`<i>` 等。

比如下面这个例子，我们想要将 **二级标题** `h2` 的字体变为 **斜体**，**绿色**，仅一个 `<h2>` 元素是不能实现的，因为它只定义了结构上的信息。

那么通过 `HTML` 提供的 `` 和 `<i>` 元素，我们可以这样实现：



```
<h2><font color="green"><i>不使用CSS</i></font></h2>
```

不使用CSS

这样做虽然能达到我们想要的效果，但是有些缺点：

- 使 `HTML` 元素变得复杂，如果要实现复杂的样式，需要在单一元素上添加大量用于显示的元素
- 不利于维护，修改样式需要找到具体的元素才能修改，结构和表现耦合在一起
- 样式无法复用，实现相同的显示效果需要编写大量重复的代码

使用就能 `CSS` 能很好地解决这些问题，让 `HTML` 只表达文档的 **结构**，而 `CSS` 表达文档的 **样式**。

同样是上面的例子，我们使用 `CSS` 来实现：



```
<h2>
```

```
    使用CSS
```

```
</h2>
```

```
h2 {  
    color:green;  
    font-style:italic;  
}
```

效果如下：



► 在线演示

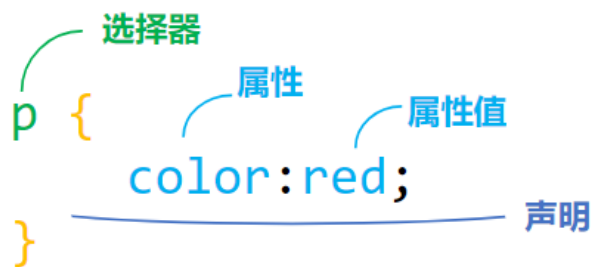
这样显示就和内容分开了，维护起来更加容易，文档的结构也清晰许多，并且样式代码能多次复用

- **CSS 语法**

了解 `CSS` 的强大易用以后，现在我们可以自己写一个 `CSS` 文件了。

`CSS` 的语法规则非常简单，不用担心写错，即使在编写 `CSS` 的过程中出现语法错误，也不会导致文档无法显示，对于不符合规则的 `CSS` 语句，浏览器会选择忽略，继续渲染下一条样式。

`CSS` 由多组 **规则** 组成，一条常见的规则如下：



- 一条规则由 **选择器 (Selector)**，大括号，和 **声明** 组成。
- 一个规则可以包含多个声明；每条声明都应该包裹在选择器后的大括号内。声明之间用 `;` 相隔。
- 一条声明内包括 **属性** 和 **属性值**，属性和属性值用 `:` 分隔。一些属性可能有多个属性值。
- `CSS` 的注释用 `/* */` 包裹。
- 如果要为多个选择器设置相同的样式，那么选择器之间用逗号 `,` 相隔，见下例

下面这条规则表示 将文档中的 **所有** `p` 元素和 `div` 元素的：

宽度设为： `80px`；

高度设为： `80px`；

文字颜色 设为： `绿色`；

背景色 设置为： `白色`；

边框 设置为： `1px`，实线， `绿色`；

可以看出 `CSS` 的属性还是比较容易理解的，比较直白，遇到生疏的属性查阅后记下来就好了。



```
p,div {  
    width: 80px;  
    height: 80px;  
    color: green;  
    background-color: white;  
    border: 1px solid green;  
}
```

效果如下：

HTML ▾	CSS ▾	Output
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width, height=device-height"> <title>JS Bin</title> </head> <body> <p> hello </p> <div> world </div> </body> </html></pre>	<pre>p,div { width: 80px; height: 80px; color: green; background-color: white; border: 1px solid green; }</pre>	

▶ 在线演示

CSS 的属性非常多，需要大家在学习的过程中不断积累，目前掌握一些常用的属性即可。

- **CSS** 的层叠概念

层叠（Cascading *[kæˈsˌkeɪdɪŋ]*）是什么意思呢？

根据 [知乎问答](#) 答主 [华南虎](#) 摘抄的 [论文](#) 片段（[Cascading Style Sheets](#) - Håkon Wium Lie）：

[Håkon Wium Lie](#) - 哈肯·维姆·莱，CSS 开发者之一，以于1994年提出的 CSS 概念而闻名（[维基百科](#)）

The process of combining several style sheets – and resolving conflicts if they occur – is known as cascading.

组合多个样式表的过程 —— 如果发生冲突，解决冲突 —— 被称为层叠。

样式引入的顺序，**选择器权重**，都会决定**样式的优先级**，优先级高的样式会覆盖优先级低的样式，**CSS** 根据样式的优先层级，对多个样式进行组合，最后应用样式，这样的处理方式就是**层叠**。

关于 **选择器** 和 **样式的优先级** 会在之后讲解。如果想要先了解，可以参见 [MDN-CSS选择器](#)，[MDN-优先级是如何计算的](#)

2. 如何使用 CSS

写完 **CSS**，需要将其应用到文档才能生效，引入 **CSS** 的方式有很多种，可以将 **CSS** 以文件形式引入，也可以在 **HTML** 中使用 `<style>` 标签引入，还可以直接将其**内联**到 **HTML** 元素中。下面将介绍引入 **CSS** 的几种方式。

1. 引入 **CSS** 的几种方式

1. 通过 `<link>` 元素在 **HTML** 的 `head` 内引入**外部** **CSS** 文件

```
<link rel="stylesheet"
href="https://www.w3school.com.cn/html/css/test1.css"/ >
```

`css/test1.css` 的 **文件** 中，定义了一条规则，将页面的 `div` 元素背景色都设置为 黑色

2. 在 **HTML** 的 `<style>` 标签中书写 **内部样式**，只对当前页面生效



```
<style>
div {
  width: 80px;
  height: 80px;
  background-color: red;
}
</style>
```

3. 在 `HTML` 元素的 `style` 属性中编写 **内联样式**, 只对当前元素生效, **权重高**, 仅次于 `!important` 修饰符



```
<div style="background-color: green;"></div>
```

这三种方式 **内联样式** 的优先级最高, 所谓 *近水楼台先得月*

当对元素应用样式产生冲突(对同一个元素由多个声明)时, 优先级高的样式会将优先级低的样式覆盖。不过, 样式的优先级由选择器的 **权重** 决定, 选择器 权重 相同的情况下, 才会考虑 样式在文档中的 顺序

下面先对 **选择器** 作介绍


2.CSS 选择器

- 选择器种类

- **通配符**

在 `CSS` 中, 一个星号 (`*`) 就是一个通配选择器, 它可以匹配任意类型的 `HTML` 元素, 这意味着它会将所有的元素进行渲染。

通配符和其他选择器搭配时, 省略掉通配选择器不会影响效果, 例如, `*.class` 与 `.class` 的效果完全相同。



```
<style>
* {
    color: red; /*所有元素的字体颜色都会设置成红色*/
}
</style>
```

○ 元素选择器

CSS 元素选择器（也叫做类型选择器）通过 node 节点名称匹配元素，在单独使用时，元素选择器会匹配文档中 **所有** 此类型的元素。



```
<style>
div {
    background-color: red;
    color: green;
}
</style>
```

上例代码，文档中 **所有的div** 的背景色会变成 红色，字体变为绿色。

○ 类选择器

在一个HTML文档中，CSS 类选择器会根据元素的**类属性**中的内容匹配元素。

类属性被定义为一个以空格分隔的列表项，在这组类名中，必须有一项与类选择器中的类名完全匹配，此条样式声明才会生效。

类选择器的语法：

```
.类名{ 样式声明 }
```


例如：

```
● ● ●  
<style>  
  .classy {  
    background-color: DodgerBlue;  
  }  
</style>  
  
<span class="classy">Here is a span with some text.</span>  
<span>Here is another.</span>
```

上面的代码产生的效果如下：

Here is a span with some text.

Here is another.

○ id 选择器

在一个 HTML 文档中，CSS ID 选择器会根据该元素的 ID 属性中的内容匹配元素，元素 ID 属性名必须与选择器中的 ID 属性名完全匹配，此条样式声明才会生效。

id选择器的语法：

```
#id属性值 { 样式声明 }
```

例如：



```
<style>
#identified {
  background-color: DodgerBlue;
}
</style>
<span id="identified">Here's a span with some text.</span>
<span>Here's another.</span>
```

产生的效果如下：

Here is a span with some text.

Here is another.

○ 伪类选择器

CSS **伪类** 是添加到选择器的关键字，指定要选择的**元素的特殊状态**。

伪类连同伪元素一起，他们允许你不仅仅是根据文档 DOM 树中的内容对元素应用样式，而且还允许你根据诸如像导航历史这样的外部因素来应用样式（例如 `:visited`）。

同样的，可以根据内容的状态（例如在一些表单元素上的 `:checked`），或者鼠标的位置（例如 `:hover` 让你知道是否鼠标在一个元素上悬浮）来应用样式。

例如：



```
<style>
    .example:hover {
        box-shadow: 0 0 10px black;
    }
</style>
<button class="example">
    点我搜索
</button>
```


效果如下：

演示效果

伪类选择器标准[索引](#)

◦ 伪元素选择器

伪元素是一个附加至选择器末的关键词，允许你对被选择元素的特定部分修改样式。例如 `::first-line` 伪元素可改变段落首行文字的样式。



```
<style>
p::first-line {
    color: blue;
    text-transform: uppercase;
}
</style>
<p>hello<br>hello</p>
```

效果如下：

HELLO

hello

伪元素选择器标准[索引](#)

- **标签属性选择器**

CSS **属性选择器**通过已经存在的属性名或属性值匹配元素。

属性选择器的语法：

```
● ● ●  
[attr]
```

表示带有以 attr 命名的属性的元素。

```
● ● ●  
<style>  
a[title] {  
  color: purple;  
}  
</style>  
/* 存在title属性的<a> 元素 */
```

```
● ● ●  
[attr=value]
```

表示带有以 attr 命名的属性，且属性值为 value 的元素。

```
● ● ●  
<style>  
a[title='abc'] {  
  color: purple;  
}  
</style>  
/* 存在title属性值为abc的<a> 元素 */
```



```
[attr~=value]
```

表示带有以 attr 命名的属性的元素，并且该属性是一个以空格作为分隔的值列表，其中至少有一个值为 value。



```
<style>
a[class~="logo"] {
  padding: 2px;
}
</style>

/* 存在class属性并且属性值包含以空格分隔的"logo"的<a>元素 */
```



```
[attr|=value]
```

表示带有以 attr 命名的属性的元素，属性值为“value”或是以“value-”为前缀（“-”为连字符，Unicode 编码为 U+002D）开头。典型的应用场景是用来匹配语言简写代码（如 zh-CN，zh-TW 可以用 zh 作为 value）。



```
[attr^=value]
```

表示带有以 attr 命名的属性，且属性值是以 value 开头的元素。



```
[attr$=value]
```

表示带有以 attr 命名的属性，且属性值是以 value 结尾的元素。



```
[attr*=value]
```

表示带有以 attr 命名的属性，且属性值至少包含一个 value 值的元素。

值得一提的是，`[id=value]` 与 id 选择器的效果相同，`[class=value]` 与类选择器的效果相同。

○ 关系选择器

关系选择器通常有四种，分别是后代选择器、子代选择器、相邻兄弟选择器、通用兄弟选择器。

首先是**后代选择器**：后代组合器（通常用单个空格（）字符表示）组合了两个选择器，如果第二个选择器匹配的元素具有与第一个选择器匹配的祖先（父母，父母的父母，父母的父母的父母等）元素，则它们将被选择。利用后代组合器的选择器称为后代选择器。



```
<style>
  #abc p {
    background-color: red;
  }
</style>
```

如以上代码，它会作用于 id 为 abc 的元素下面所有的 p 元素，这些 p 元素的背景都是红色。

然后是**子代选择器**，当使用 `>` 选择符分隔两个元素时，它只会匹配那些作为第一个元素的**直接后代**(子元素)的第二元素。与之相比，当两个元素由后代选择器相连时，它表示匹配存在的所有由第一个元素作为祖先元素(但不一定是父元素)的第二元素，无论它在 DOM 中"跳跃"多少次。

例子：



```

<style>
span { background-color: white; }
div > span {
  background-color: DodgerBlue;
}
</style>
<div>
  <span>Span 1. In the div.
    <span>Span 2. In the span that's in the div.</span>
  </span>
</div>
<span>Span 3. Not in a div at all</span>


```

效果展示：

Span 1. In the div. Span 2. In the span that's in the div.

Span 3. Not in a div at all.

第三个是**相邻兄弟选择器**，相邻兄弟选择器 (`+`) 介于两个选择器之间，当第二个元素紧跟在第一个元素之后，并且两个元素都是属于同一个父元素的子元素，则第二个元素将被选中。



```

<style>
  button + p {
    color: red;
  }
</style>
<div>
  <button>
    button1
  </button>
  <p>
    p1
  </p>
  <p>

```

```

        p2
      </p>
    <button>
      button2
    </button>
  <p>
    p3
  </p>
</div>

```

效果如下：

button1

p1

p2

button2

p3

最后是**通用兄弟选择器**，位置无须紧邻，只须同层级，`A~B` 选择 `A` 元素之后所有同层级 `B` 元素。

```

● ● ●
<style>
  button ~ p {
    color: red;
  }
</style>
<div>
  <button>
    button1
  </button>
  <p>
    p1
  </p>
  <p>

```



```
        p2
    </p>
    <p>
        p3
    </p>
</div>
```

效果如下：

button1

p1

p2

p3

• 样式的继承

一些设置在父元素上的css属性是可以被子元素继承的，有些则不能。举一个例子，如果你设置一个元素的 `color` 和 `font-family` ，每个在里面的元素也都会有相同的属性，除非你直接在元素上设置属性。

继承的样式没有优先级

一些属性是不能继承的 — 举个例子如果你在一个元素上设置 `width` 50% ，所有的后代不会是父元素的宽度的50%。

至于哪些属性能够继承，哪些不能够继承多是由常识决定的。

有些元素是能够继承父元素的样式的，这种类型被称作**自然继承**。

哪些子元素是自然继承的而哪些不是也只能依靠经验的积累，这里不再赘述。

`CSS` 为控制继承提供了四个特殊的通用属性值。每个 `css` 属性都接收这些值。

- `inherit`

设置该属性会使子元素属性和父元素相同。实际上，就是 "开启继承"。

- `initial`

设置属性值和浏览器默认样式相同。如果浏览器默认样式中未设置且该属性是自然继承的，那么会设置为 `inherit` 。

- `unset`

将属性重置为自然值，也就是如果属性是自然继承那么就是 `inherit`，否则和 `initial` 一样

- `revert`

这个属性只有很少的浏览器支持，不详细说明。

```

<style>
body {
    color: green;
}

.my-class-1 a {
    color: inherit;
}

.my-class-2 a {
    color: initial;
}

.my-class-3 a {
    color: unset;
}
</style>
<ul>
    <li>Default <a href="#">link</a> color</li>
    <li class="my-class-1">Inherit the <a href="#">link</a>
color</li>
    <li class="my-class-2">Reset the <a href="#">link</a>
color</li>
    <li class="my-class-3">Unset the <a href="#">link</a>
color</li>
</ul>
```

效果如下：

展示

- 选择器权重

千位： 如果声明在 `style` 的属性（内联样式）则该位得一分。这样的声明没有选择器，所以它得分总是 `1000`。

百位： 选择器中包含 `ID` 选择器则该位得一分。

十位： 选择器中包含类选择器、属性选择器或者伪类则该位得一分。

个位： 选择器中包含元素、伪元素选择器则该位得一分。

在进行计算时**不允许进行进位**，例如，20 个类选择器仅仅意味着 20 个十位，而不能视为 两个百位，也就是说，无论多少个类选择器的权重叠加，都不会超过一个 ID 选择器

```
● ● ●
<style>
/* specificity: 0101 */
#outer a {
    background-color: red;
}

/* specificity: 0201 */
#outer #inner a {
    background-color: blue;
}

/* specificity: 0104 */
#outer div ul li a {
    color: yellow;
}

/* specificity: 0113 */
#outer div ul .nav a {
    color: white;
}
```

```
/* specificity: 0024 */
div div li:nth-child(2) a:hover {
    border: 10px solid black;
}

/* specificity: 0023 */
div li:nth-child(2) a:hover {
    border: 10px dashed black;
}

</style>
```

综上：样式的优先级排序

权重高的样式会覆盖掉权重低的样式，当权重相同的情况下，后引入的样式会覆盖前面引入的样式。

另外的，有一个特殊的 `CSS` 可以覆盖所有的优先级计算——`!important`，`!important` 总是会覆盖掉其他样式，除了另一个 `!important` 具有相同优先级而且顺序靠后，或者更高优先级。

3.浏览器的默认样式

通过上面所提到几种方式引入的样式都称为 `用户样式`，除了用户样式，浏览器内置了一种默认样式，称为 `用户代理样式 (user agent stylesheet)`，以下都称为 **默认样式**（浏览器也被称为 `用户代理`）

- 默认样式的作用：

默认样式保证了网页的基本排版，当元素没有指定的样式时，浏览器会按照默认样式来渲染元素。

比如，一级标题通常是大号加粗字体且独占一行，二级标题也独占一行但字号会小一点，还比如列表的样式，在每项前添加圆点，以表示这些文字不是独立存在的。

```
h1 {                                user agent stylesheet
  display: block;
  font-size: 2em;
  margin-block-start: 0.67em;
  margin-block-end: 0.67em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  font-weight: bold;
}
```

chrome浏览器一级标题的默认样式(可以在开发者工具中查看)

```
ul {                                user agent stylesheet
  display: block;
  list-style-type: disc;
  margin-block-start: 1em;
  margin-block-end: 1em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  padding-inline-start: 40px;
}

li {                                user agent stylesheet
  display: list-item;
  text-align: -webkit-match-parent;
}

Inherited from ul
ul {                                user agent stylesheet
  list-style-type: disc;
}
```

chrome浏览器列表的默认样式

►在线演示

浏览器的默认样式对这些都有预设,如果用户不满意,可以自己写样式覆盖掉。

[🔗 查看 webkit 内核浏览器默认样式](#)

- 默认样式带来的问题:

不同的浏览器的默认样式不一样,甚至同一个浏览器,版本不同,默认样式也会有区别,所以同一个网页在不同的浏览器中会呈现出不同的效果

(并且每个浏览器对 [CSS规范](#) 的实现有差异,也会导致网页的表现不一样,通常需
要解决浏览器的 兼容性问题 [MDN-常见的跨浏览器问题](#) 对这方面有一些说明,可
参考)

并且,有时候浏览器的默认样式,我们不满意,比如 `chrome` 的默认样式会给 `body` 元素添加 `8px` 的外边距:



- 消除默认样式:

可以针对某一个样式,自己写一条规则覆盖掉默认样式

引入 `normalize.css` 对样式进行统一

4.项目中对 `CSS` 的处理

- 预处理器 如 `less`

对 `CSS` 进行了一些扩展,使用 `嵌套` 的方式书写代码, `&` (and 符号) 表示父元素,代码更加简洁,还提供了 `变量` 和 `函数` 等其他功能

- 遵循样式书写规范,包括选择器命名,属性书写顺序等。
 - 按照团队约定规范执行

3. `CSS` 基础内容

1.盒模型

在 `CSS` 中,所有的元素都被一个个的“盒子 (box)”包围着 (`MDN-盒模型`)

`CSS` 中将盒子分为 `块级盒子` 和 `内联盒子`,他们在页面上的表现不一样

- 块级盒子(`display:block`) 有以下特点
 - 默认宽度撑满父元素 ▶[在线演示](#)

- 独占一行显示 [▶在线演示](#)
- 设置宽(`width`) 高(`height`)有效
- 内边距 (`padding`), 外边距 (`margin`) 和 边框 (`border`) 会将其他元素从当前盒子周围 “推开”
- 内联盒子(`display:inline`) 有以下特点
 - 不会独占一行
 - 设置宽(`width`) 高(`height`)**无效**
 - 垂直方向的内边距、外边距以及边框 会被应用但是不会推开其他盒子 [▶在线演示](#)
 - 水平方向的内边距、外边距以及边框会被应用且会把其他盒子推开。

通过 `CSS` 的 `display` 属性, 可以改变盒子的**外部**显示类型(在其他盒子中怎样显示)

将 `display` 设置为 `inline-block` 可以让盒子不独占一行, 同时宽高的设置也有效,相当于结合了块级盒子和内联盒子的特点.

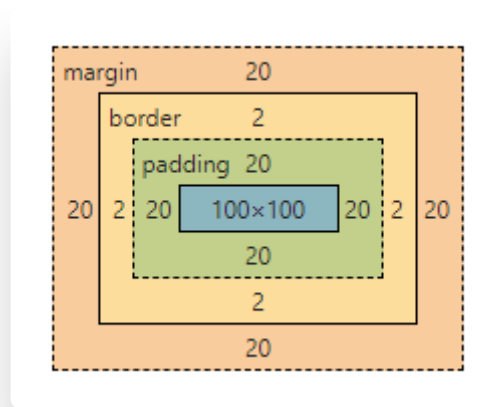
- 盒模型

标准盒模型,盒子的宽和高 **不包括** 外边距, 边框, 内边距

页面上渲染的盒子的大小包括 :内容大小(也就是宽高属性设置的大小), 内边距,边框,外边距

以一个块级盒子为例 [▶在线演示](#)

```
div {  
  display: block;  
  height: 100px;  
  width: 100px;  
  background-color: black;  
  color: white;  
  border: 2px solid red;  
  margin: 20px;  
  padding: 20px;  
}
```



标准盒模型

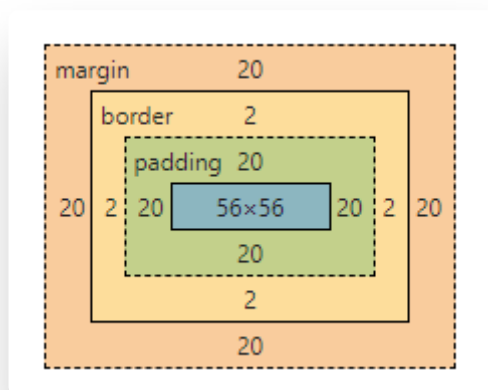
替代(IE)盒模型 盒子的宽和高 **包括** 盒子内容，边框和内边距,实际**盒子内容**的宽高还要减去边框和内边距（不包括外边距）

目前绝大多数浏览器默认采用的是 **标准盒模型** (`box-sizing : content-box;`)

IE浏览器 默认使用替代盒模型，没有可用的机制来切换。（IE8+ 支持使用 `box-sizing` 进行切换 - [MDN](#)）

将 `box-sizing` 属性设置为 `border-box` 可以将盒子模型变为 替代盒模型

► [在线演示](#)



替代盒模型

2.文档流

- 正常布局流

浏览器默认的布局方式，页面上的元素按照 `HTML` 文档中的先后顺序出现

►在线演示

当正常流布局不能满足我们的要求时，我们可以通过一些 **布局技术** 来实现想要的效果，比如：

- 设置 `display` 属性为 `block` , `inline` 或 `inline-block` 来改变布局
- 开启浮动 `float`
- `flex` 布局
- `grid` 布局（见 [MDN-网格](#)）
- 定位

其中 开启浮动 和 绝对定位 会使元素 脱离文档流，也就是说它所占用的空间会被忽略，其他元素在布局时不会考虑它，关于脱离文档流，可以参考 [知乎问答-脱离文档流](#)

`float` 浮动及清除浮动可以参见 [MDN-float](#), 下面主要对 `flex` 布局 和 `定位` 作介绍。

3.flex 布局

弹性盒子布局：一维页面布局，弹性布局使得页面布局变得更加容易，如居中子元素，等量分配子项空间等。

- 弹性容器(盒子)

通过将（块级）盒子的 `display` 属性设置为 `flex` , 可以将盒子变为弹性容器。

假如你想设置行内元素为 flexible box，也可以置 `display` 属性的值为 `inline-flex` ([--MDN](#))

弹性容器具有以下几个特点：

- 子元素自动成为弹性项 (`flex item`)
- 容器存在两根轴，主轴 (`main axis`) 和 交叉轴 (`cross axis`), 两轴相互垂直。
- 弹性项默认沿主轴排列，主轴默认为水平方向。

►在线演示

通过设置弹性容器的属性，可以决定弹性容器中弹性项的排列方式

1. 弹性项排列的方向（即主轴的方向） `flex-direction`

- `row`（默认值）：主轴为水平方向，从左至右
- `row-reverse`：主轴为水平方向，从右至左。
- `column`：主轴为垂直方向，从上到下。
- `column-reverse`：主轴为垂直方向，从下至上。

►在线演示

2. 弹性项是否换行展示

默认情况下，项目都排列在一条轴线上，如果一条轴线放不下，那么通过 `flex-wrap` 属性来决定是否换行显示。

- `nowrap`（默认）：不换行。弹性项的大小会被压缩，根据项目原本的大小，按比例分配空间 ►在线演示
- `wrap`：换行，从上到下排列。
- `wrap-reverse`：换行，从下至上排列。

3. 弹性项对齐方式 & 空间分布（摘自：[阮一峰-Flex布局教程：语法篇](#)）

`justify-content` 属性定义了项目在**主轴**上的对齐方式。

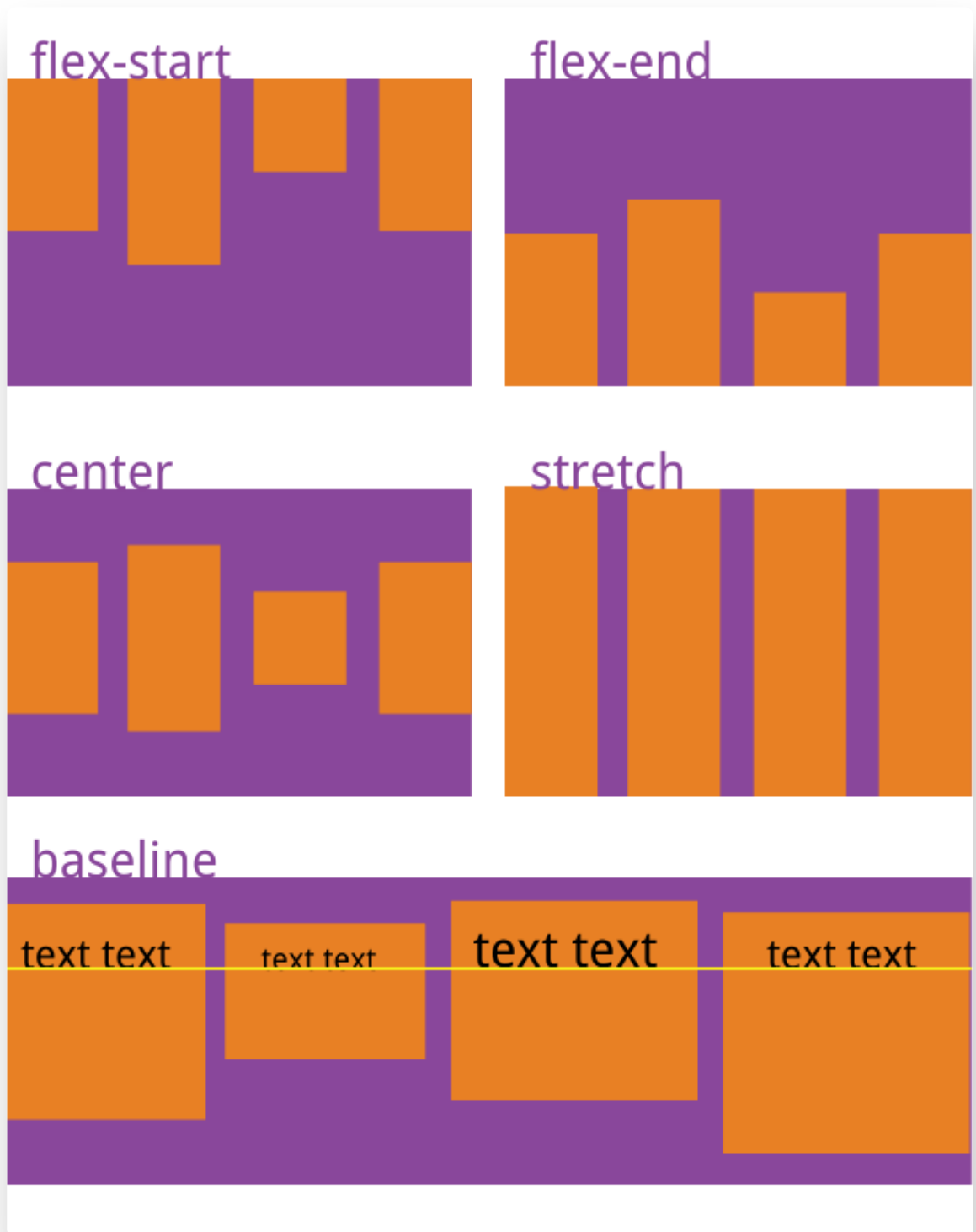
- `flex-start`（默认值）：左对齐
- `flex-end`：右对齐
- `center`：居中
- `space-between`：两端对齐，项目之间的间隔都相等。
- `space-around`：每个项目两侧的间隔相等。

►在线演示

`align-items` 属性定义项目在**交叉轴**（与主轴垂直）上如何对齐。

- `stretch`（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度。
- `flex-start`：交叉轴的起点对齐。

- `flex-end` : 交叉轴的终点对齐。
- `center` : 交叉轴的中点对齐。 ▶ [在线演示](#)
- `baseline` : 项目的第一行文字的基线对齐。



图片：交叉轴对齐方式 - [阮一峰-Flex布局教程：语法篇](#)

- 弹性项

弹性容器的子元素称为弹性项，除了对弹性容器的属性进行设置来改变弹性布局，也可以对每个弹性项单独配置属性，来控制弹性项的顺序，弹性系数等

以下内容摘自：[阮一峰-Flex布局教程：语法篇](#)（有部分修改）

- 弹性项的顺序 [▶在线演示](#)

`order` 属性定义项目的排列顺序。数值越小，排列越靠前，默认为 `0`。

- 弹性项的放大比例 [▶在线演示](#)

`flex-grow` 属性定义项目的放大比例，默认为 `0`，即如果存在剩余空间，也不放大。

如果所有项目的 `flex-grow` 属性都为1，则它们将等分剩余空间（如果有的话），如果一个项目的 `flex-grow` 属性为 `1`，其他项目都为 `0`，则前者将占据剩余所有空间。

- 弹性项的缩小比例

`flex-shrink` 属性定义了项目的缩小比例，默认为 `1`，即如果空间不足，该项目将缩小。

如果所有项目的 `flex-shrink` 属性都为1，当空间不足时，都将等比例缩小。如果一个项目的 `flex-shrink` 属性为0，其他项目都为1，则空间不足时，前者不缩小。



图片：`flex-shrink: 0` [阮一峰-Flex布局教程：语法篇](#)

- 弹性项初始占据空间

`flex-basis` 属性定义了再分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为 `auto`，即项目的本来大小。

- 简写属性 `flex`

`flex` 属性是 `flex-grow`，`flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto`。后两个属性可选。

`flex` 实例可参考 [Flex 布局教程：实例篇](#)

目前大多数浏览器（如 Firefox, Chrome, Opera, Microsoft Edge 和 IE 11）都支持 `flex` 布局，不过如果你要在比较老的浏览器使用 `flex`，那需要考虑其兼容性问题。

4.定位

通过定位，可以将元素置于文档中的任意位置，设置 `position` 属性来决定元素的定位模式。

注：以下关于元素定位的原点（即元素相对于什么位置进行偏移）不太准确，具体可以参见 [MDN-布局 and 包含块](#)

- 静态定位

`position:static;` 元素的默认定位,正常布局流，下文中 开启定位，都指 非静态定位的情况

- 相对定位

`position: relative;`

相对于元素的默认位置进行偏移，通过设置 `top`，`left`，`right`，`bottom` 来指定元素的位置。

开启相对定位后，如果元素之间重叠，则开启相对定位的元素会覆盖静态定位的元素。

► [在线演示](#)

- 绝对定位

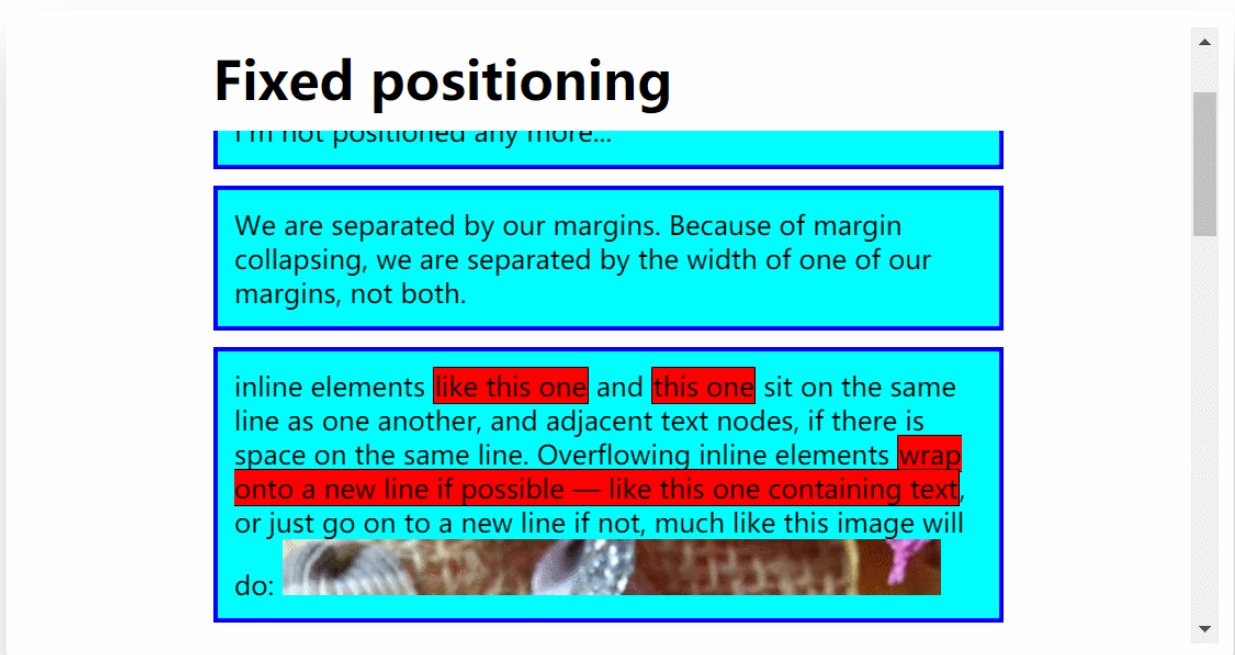
```
position: absolute;
```

开启绝对定位的元素会从正常布局中脱离，即脱离文档流，元素相对于其**包含块**定位，包含块是离该元素最近的开启了 `relative`，`absolute`，`fixed` 或 `sticky` 定位的**祖先**元素，如果没有开启了这些定位的祖先元素，那么其包含块为**初始块容器**。（简单记作元素根据页面最左上角定位就行了）

► 在线演示

- 固定定位

`position: fixed;`，固定可以看作是一种特殊的绝对定位，不过，固定定位的元素的位置是相对于浏览器**视口**的，下面是一个使用固定定位固定页面顶部标题的[例子-MDN](#)



- 粘滞定位

```
position: sticky;
```

粘滞定位，表现得和相对定位的元素一样，不过当它滚动到页面某个特定的位置时（如视口顶部），它就固定在那个位置了，下面是一个粘滞定位的[例子-MDN](#)，当粘滞项滚动到视口顶部，就会固定在那。

Sticky positioning

A

Apple
Ant

- 元素的层级

开启定位后，元素之间可能会发生重叠，如下：



如果想要改变层叠的顺序，可以通过设置 `z-index` 的属性值实现，值越大，层级越高。

祖先元素的层级再高也不会覆盖其子元素 [►在线演示](#)

网页也有一个z轴：一条从屏幕表面到你的脸（或者在屏幕前面你喜欢的任何其他东西）的虚线。

`z-index` 值影响定位元素位于该轴上的位置；正值将它们向上移动，负值将它们向下移动。默认情况下，定位的元素都具有 `z-index` 为 `auto`，实际上为 `0`（MDN）

5.单位

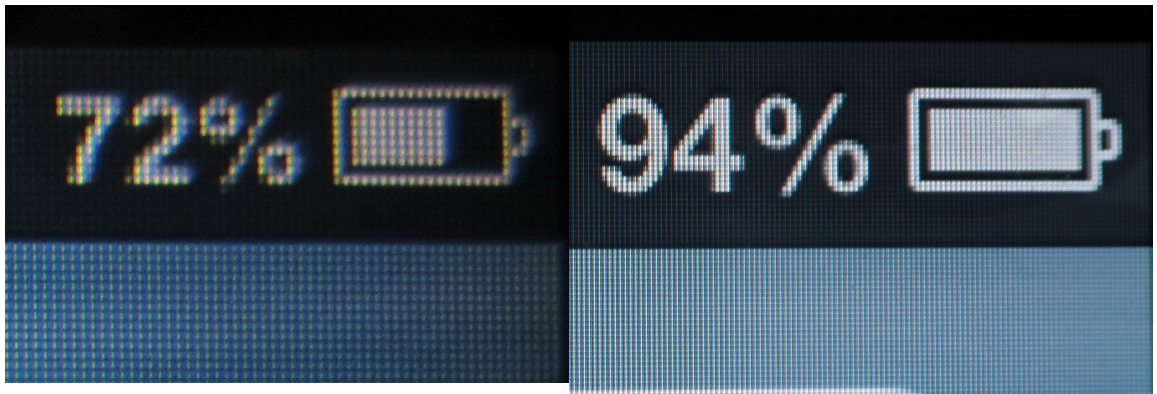
- 像素

- 物理像素（设备像素 `DP`）

显示屏的**实际**像素点(`point`), 最小显示单位, 通过控制每个像素点的颜色, 呈现出整个图像。

不同的设备所拥有的像素点数是不同的。单位面积内的像素点越多 (`ppi` 越高), 图像越清晰。

如下 `iPhone3GS` 与 `iphone4` 屏幕的对比, 它们的屏幕尺寸相同, 不过 `iphone4` 的像素密度更大, 所以图像也更清晰。



图片: *iPhone3Gs-Non-Retina* 作者:
Haotian0905

图片: *iPhone4-Retina* 作者:
Haotian0905

◦ 设备独立像素

为什么不使用物理像素作为单位?

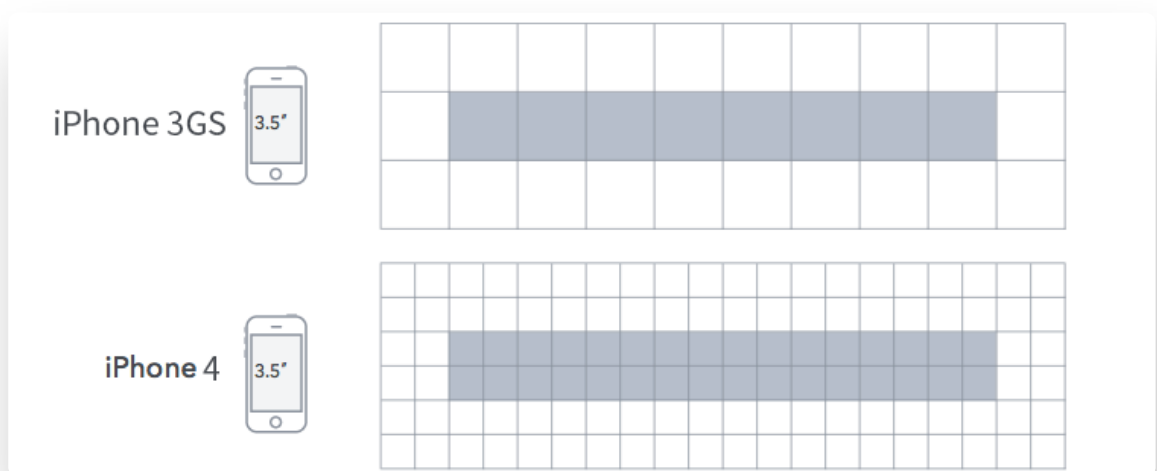
例: `iPhone 3` 和 `iPhone 4`, 屏幕尺寸一样, 后者的像素密度是前者的两倍, 如果使用物理像素作为单位, 那么 `div` 的**宽度**视觉上是 `iPhone 4` 是 `iPhone 3` 的一半, 所以使用物理像素在 `web` 中设置尺寸是不合理的。

因此操作系统定义设备独立像素, 用设备独立像素定义的尺寸, 不管屏幕的参数如何, 都能以合适的大小显示。

iPhone 3 物理像素是 320×480 ，iPhone 4 的物理像素是 640×960 ，但是它们的设备独立像素都是 320×480 ，iPhone 4 用 4 个物理像素表示 1 个设备独立像素

(理解CSS像素，设备像素和设备独立像素 -oWSQo)

我们平时描述元素的尺寸所使用的单位 px ，是一个抽象单位，默认情况下（页面未缩放）， 1px 所描述的就是一个设备独立像素。



图片：相同的设备独立像素在 iPhone3GS 与 iPhone4 的对比

图源

○ DPR 设备像素比

即 物理像素 / 设备独立像素，比如提到 iPhone 6 物理像素为 750×1334 ，而它的设备独立像素是 375×667 ，那么它的 $\text{dpr} = 750 / 375 = 2$

iPhone 6/7/8 ▼ 375 × 667

也就是说，在 iPhone 6 上，我们 CSS 写的一个 $1\text{px} \times 1\text{px}$ 的小方块，实际所占据的物理像素为 2×2 也就是 4 个物理像素点。

- 绝对长度

物理长度单位，如 `cm`，`mm` 等，打印时比较常用。

- `em`，`rem`

`em`，`rem` 是相对单位

- `1em` = `1 font-size` (父元素的字体大小)
- `1rem` = 1 根元素字体大小，根元素即 `<html>` 元素

使用 `rem` 作为单位，随后搭配 媒体查询 或 `JS`，根据屏幕的大小来**动态控制** `html` 元素的 `font-size`，即可自动改变所有用 `rem` 定义尺寸的元素的大小。

使用 `rem` 实现 `flexible` 布局，参见 [使用Flexible实现手淘H5页面的终端适配](#)

- 百分比 `%`

元素的大小不是固定的，根据百分比，随着其父元素的大小变化而变化。

- 视口单位，`vw`，`vh`

`vw`：视口宽度

`vh`：视口高度

- 颜色单位

- `rgb` - 红，绿，蓝
- `rgba`： `a` 表示不透明度 从 `0 -1`，`0` 表示完全透明
- `HSL`，`HSLA`
 - `H` 色相 (0-360)
 - `S` 饱和度 (颜色浓度) 0%-100%
 - `L` 亮度 0%-100%
 - `A` 不透明度

6.视口

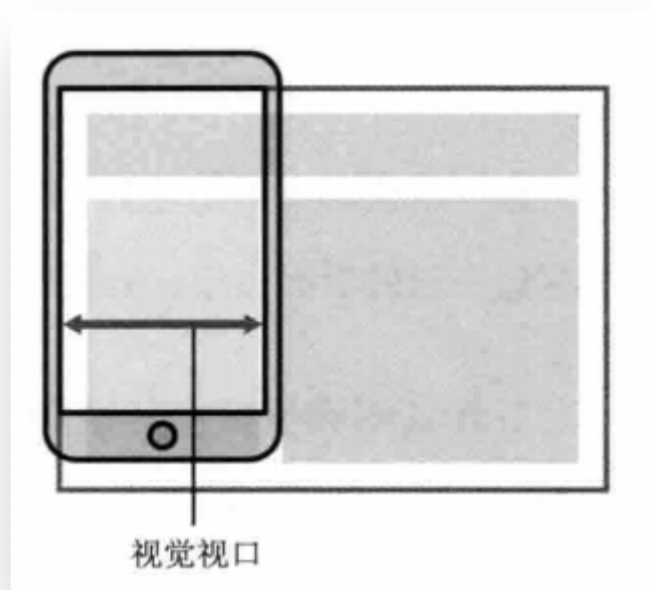
视口 (`viewport`) 通常与浏览器窗口相同，但不包括浏览器的 UI， 菜单栏等——即指你正在浏览的文档的那一部分

概括地说， `viewport` 基本上是当前文档的可见部分。([MDN-视口概念](#))

- 移动端视口
 - 布局视口

因为常用的移动设备（如手机）屏幕太小，为了让桌面浏览器上的网页在手机上也能正常显示，移动端网页的视口默认为 `980px`，用户通过缩放，滚动等方式查看网页。

- 视觉视口



视觉视口，[图源](#)

- 理想视口

因为现在大多数网页会对移动端做适配，为了使网页在移动端能以最好的效果展示， `Apple` 和其他浏览器厂商引入了理想视口的概念，它对设备而言是最理想的布局视口尺寸。显示在理想视口中的网站具有最理想的宽度，用户无需进行缩放。

通常来说，理想视口的宽度就是设备的宽度，所以通常使用下面这行代码，将视口的宽度设置为和设备宽度一致。



```
<meta name="viewport" content="width=device-width,initial-scale=1.0">
```

关于 移动端视口 可参考 [浅谈移动端中的视口 \(viewport\)](#)

- `vw` , `vh`

`1vw` = `1%` 视口宽度;

`1vh` = `1%` 视口高度

7.响应式

- 传统布局

统一使用 `px` 作为单位，页面布局固定，浏览器窗口缩小时，被遮挡部分通过滚动条浏览。

移动端另外搭建网站，采用不同布局。

- 流式布局

使用 `%` 作为宽度单位，高度，字体大小等通常使用 `px` 进行固定，元素的宽度随着窗口大小改变而伸缩，

但是页面布局不会改变。

- 响应式设计

响应式设计的目标是确保一个页面在所有终端上，都能显示出令人满意的效果（[静态布局](#)、[自适应布局](#)、[流式布局](#)、[响应式布局](#)、[弹性布局](#)等的概念和区别)，通常会结合流式布局，`flex` 布局，媒体查询等多种技术实现页面随着窗口大小的改变而伸缩，调整布局以达到较好的显示效果。

- 媒体查询 - 有条件的应用样式

CSS媒体查询为你提供了一种应用CSS的方法，仅在浏览器和设备的环境与你指定的**规则相匹配**的时候CSS才会真的被应用（MDN）

- 语法

每条媒体查询语句都由一个可选的 *媒体类型* 和 任意数量的 *媒体特性* 表达式构成。

如：

```
● ● ●
@media screen and (min-width: 680px) and (max-width:
980px) {
    body {
        background-color: green;
    }
} /*屏幕上 680px 至 980px 之间, 将 body 元素的背景色设置为绿色*/
```

关于媒体查询，参见 [MDN-媒体查询](#)

4.CSS3 新特性

可以参考 [segmentfault - 个人总结 \(css3新特性\)](#)

- 过渡

`transition` 过渡效果

`transition` 属性是一个简写属性，`4` 个值分别对应以下几个 属性

`transition: width 1s ease 0.5s;`

```
● ● ●
transition-property: width; /* 要进行过渡的属性 */
transition-duration: 1s; /* 过渡持续的时间 */
transition-timing-function: ease; /* 时序函数 ， 动画的速度曲线，可由
cubic-bezier() 函数指定*/
transition-delay: 0.5s; /* 动画延迟执行的时间 */
```

► [在线演示](#)

下面是一个 [MDN](#)的示例

可以到 cubic-bezier.com 生成想要的贝塞尔曲线，常用的有 5 种预设，默认值是 `ease` 可以到上述网站查看效果



- 变形

`transform` 参见 [MDN-transform-function](#)

- 旋转 `rotate()`
- 平移 `translate()` [▶在线演示](#)
- 缩放 `scale()`

- 阴影

`box-shadow`

语法 ([MDN](#))

```
/* x偏移量 / y偏移量 / 阴影颜色 */
box-shadow: 60px -16px teal;

/* x偏移量 / y偏移量 / 阴影模糊半径 / 阴影颜色 */
box-shadow: 10px 5px 5px black;

/* x偏移量 / y偏移量 / 阴影模糊半径 / 阴影扩散半径 / 阴影颜色 */
box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);
```

- 边框圆角

`border-radius`

border-radius还可以用 **斜杠** 设置第二组值。这时，第一组值表示水平半径，第二组值表示垂直半径。第二组值也可以同时设置1到4个值，应用规则与第一组值相同([CSS3圆角详解](#))

- 计算函数, css变量

calc()函数，支持数值的 加减乘除，如宽度，高度，颜色等

```
width: calc(100% - 80px);
```

CSS 变量

使用 `--` 声明变量，如：

```
body {  
  --textColor: #000;  
  --navHeight: 60px;  
}
```

使用 `var()` 函数读取变量，如：

```
.nav {  
  color: var(--textColor);  
  height: var(--navHeight);  
}
```

关于 `CSS变量` 可参考：

- [MDN-使用CSS自定义属性](#)
- [阮一峰-CSS 变量教程](#)

5.CSS如何运作

参见：

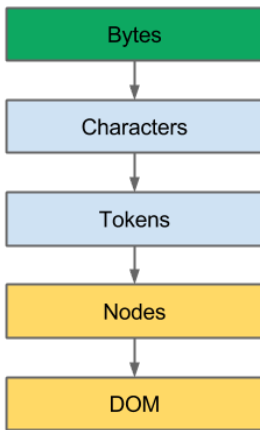
- [浏览器如何渲染 HTML & CSS](#)
- [渲染页面：浏览器的工作原理](#)
- [关键渲染路径](#)
- [从css到页面样式渲染](#)

大概流程：

- 处理 HTML 标记并构建 `DOM` 树
原始字节 --> 字符 --> 令牌化 --> 词法分析 --> `DOM` 构建
- 处理 `CSS` 标记并构建 `CSSOM` 树
- 将 `DOM` 与 `CSSOM` 合并成一个渲染树

注：渲染树只包含渲染网页所需的节点，如 `display: none;` 的元素是不会出现在渲染树中的。

- 根据渲染树来布局，计算每个节点的几何信息，再将各个节点绘制到屏幕上
 - 布局
 - 绘制
 - 重排
 - 重绘

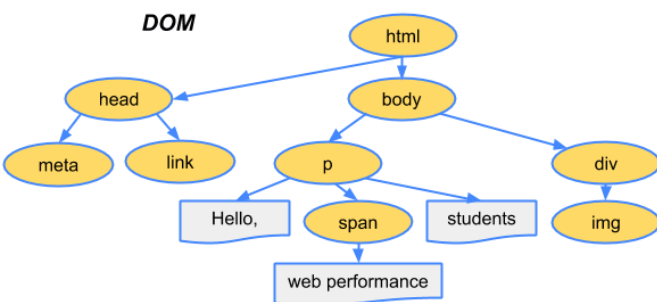
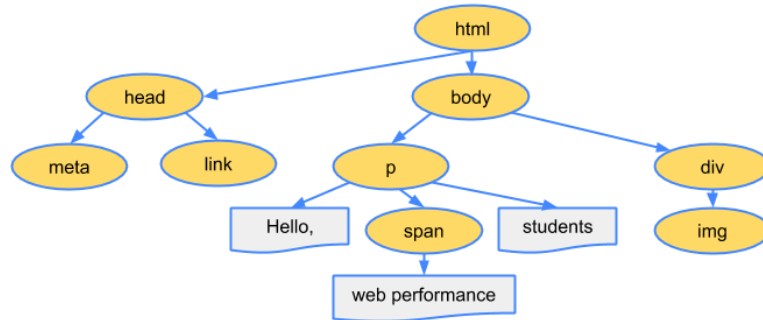


3C 62 6F 64 79 3E 48 65 6C 6C 6F 2C 20 3C 73 70 61 6E 3E 77 6F 72 6C 64 21 3C 2F 73 70 61 6E 3E 3C 2F 62 6F 64 79 3E

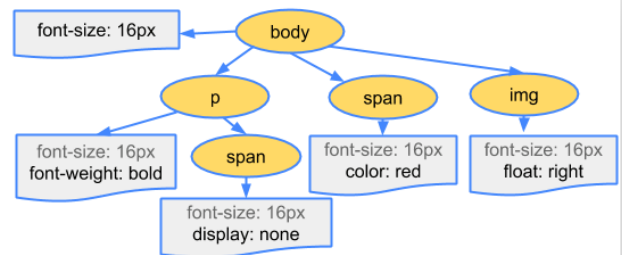
<html><head>...</head><body><p>Hello web performance...</body></html>

StartTag: html StartTag: head ... EndTag: head StartTag: body StartTag: p Hello ...

html head meta body p Hello



CSSOM



Render Tree

