CSS自存笔记

自存一些语法：

基础语法：

```
h1 ,
li{
  color: red;
  font-size: 2.5em;
}
```

`h1 ,li{ ... }`：表示选中所有的<h1> 和 <li> 标题标签。

`color: red;`：把<h1>标题的文字颜色设置为红色。

`font-size: 2.5em;`：设置字体大小为2.5倍的基础字体大小（em 是相对单位，1em 表示当前元素的字体大小，通常继承自父元素）

```
p {
  color: aqua;
  padding: 5px;
  background: midnightblue;
}
```

`p { ... }`：表示选中所有的段落<p>标签。

`color: aqua;`：设置段落文字颜色为浅蓝色（aqua）。

`padding: 5px;`：段落内容与边框之间的内边距为5像素。

`background: midnightblue;`：背景颜色设置为深蓝色。

连接HTML：

为了连接HTML和CSS，要在HTML的head中加入这句话：

```
<link rel="stylesheet" href="styles.css" />
```

以上的CSS文件和HTML文件放在同一文件夹内，而如果CSS想放置在别的位置，也是可行的：

```
<!-- In a subdirectory called styles in the current directory -->
<link rel="stylesheet" href="styles/style.css" />

<!-- In a subdirectory called general, which is in a subdirectory called styles, in the current directory -->
<link rel="stylesheet" href="styles/general/style.css" />

<!-- Go back one directory level, then in a subdirectory called styles -->
<link rel="stylesheet" href="../styles/style.css" />
```

也可以直接把CSS内容放在HTML文件的head中：

```
<style>
  p {
    color: purple;
  }
</style>
```

还可以用内联样式：

```
<span style="color: purple; font-weight: bold">span element</span>
```

用 语句，在其 style 中直接放CSS代码，此次格式的更改只对 中间的内容有用（这种方法要少用，容易引起混乱）。

补充， 是一个非常常用的**行内元素**，它的特点是

1. 不会自动换行；
2. 本身没有特殊的语义（不像 <p> 是段落，<h1> 是标题）；
3. 通常用来对**一小段文字或内容**加样式，比如改颜色、加粗、加背景色等等。

怎么只修改一个Class里面的格式?

注意到，我们上面的写法是会把找到的所有<p>类似的形式全部修改为目标格式，但是如果我只想修改一个 class 里的格式，我们可以：

先在html中设置class：

```
<ul>
  <li>Item one</li>
  <li class="special">Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

然后在CSS文件中：

```
.special {
  color: orange;
  font-weight: bold;
}
```

其中，.special 是一个 CSS 选择器，用来选中 class 名为 special 的元素。

修改特定的位置：

```
li em {
  color: rebeccapurple;
}
```

这就是只修改了li 中 em 中内容的颜色。

```
h1 + p {
  font-size: 200%;
}
```

`h1 + p` 表示，紧跟在 `<h1>` 后面的那个 `<p>` 元素。

补充，`font-size: 200%;`：设置这个 `<p>` 元素的字体大小为当前字体的两倍（200%）。

和超链接相关的：

```css
a:link {
  color: pink;
}

a:visited {
  color: green;
}

a:hover {
  text-decoration: none;
}
```

`a:link` 是指**还没点击过的超链接**（即“未访问”状态）。

`a:visited` 是指**用户点击过、访问过的链接**（即“已访问”状态）。

`a:hover`：表示鼠标**悬停在 `<a>` 超链接元素上**时的状态。

好的，接下来来看看超级加倍：

```css
h1 + p .special {
  color: yellow;
  background-color: black;
  padding: 5px;
}
```

选中 class 为 `special` 的元素，但只有当它满足以下结构时才会被选中：

- 一个 `<h1>` 元素，
- 后面紧跟着一个 `<p>` 元素，
- 而 class 为 `special` 的元素要在这个 `<p>` 元素**内部**。

补充，padding: 5px;：内边距为 5px，让背景和文字之间有间距，就是让文字四边框有留白效果。

我们神奇的CSS还可以放公式（calc）哦!

```
<div class="outer"><div class="box">The inner box is 90% - 30px.
</div></div>
```

```
.outer {
  border: 5px solid black;
}

.box {
  padding: 10px;
  width: calc(90% - 30px);
  background-color: rebeccapurple;
  color: white;
}
```

这两段代码，先:

给 .outer 添加了一个 **5px 黑色边框**，让你能看到这个容器的边界。

但它没有设置宽度或高度，所以它的大小由内容决定。

再

padding: 10px;：给内容周围留出 10px 的内边距（四边）。

width: calc(90% - 30px);：用 calc() 来计算宽度：先取父元素(就是.outer) 宽度的 90%，再减去 30 像素(pixels)。

background-color: rebeccapurple;：背景设为一种紫色。

color: white;： 文字颜色设为白色。

还有神奇的Transform函数!

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

```
.box {
  margin: 30px;
  width: 100px;
  height: 100px;
  background-color: rebeccapurple;
  transform: rotate(0.8turn);
}
```

属性	作用
`margin: 30px;`	给这个盒子四周加 30px 的外边距，让它和其他元素之间留出空隙
`width: 100px;`	设置宽度为 100 像素
`height: 100px;`	设置高度为 100 像素，形成一个正方形
`background-color: rebeccapurple;`	设置背景颜色为“rebeccapurple”（一种紫色）
`transform: rotate(0.8turn);`	让这个盒子**旋转 0.8 圈（等于 288°）**

补充，关于 `rotate(0.8turn)`:

- `1turn` 表示 360 度;
- `0.8turn = 0.8 × 360° = 288°`;
- 所以这个方块会**顺时针旋转 288 度**，看起来会歪斜。

你也可以用其他单位来旋转，比如：

- `rotate(45deg)` 表示 45 度；
- `rotate(1rad)` 表示 1 弧度；
- `rotate(0.5turn)` 表示旋转 180 度。

@rules：

CSS 中的 @rules（读作“at-rules”）是用来告诉浏览器“怎么处理 CSS”的指令。一种常见的 @rule 是 `@media`，它用于创建“媒体查询”，通过条件逻辑让网页在不同设备（如手机、电脑）上显示不同样式。

```
body {
  background-color: pink;
}

@media (min-width: 30em) {
  body {
    background-color: blue;
  }
}

```

默认背景是粉色，但如果浏览器的宽度超过 `30em`（大约480px），背景就会变成蓝色。

Shorthand Properties（简写属性）

像 `font`、`background`、`padding`、`border` 和 `margin` 这些属性叫做“简写属性”，因为它们可以用一行同时设置多个值。

例如：

```
padding: 10px 15px 15px 5px;
```

等价于：

```
padding-top: 10px;
padding-right: 15px;
padding-bottom: 15px;
padding-left: 5px;
```

顺序是：**上 → 右 → 下 → 左**（顺时针）

在比如：

```
background: red url(bg-graphic.png) 10px 10px repeat-x fixed;
```

等价于：

```
background-color: red;
background-image: url(bg-graphic.png);
background-position: 10px 10px;
background-repeat: repeat-x;
background-attachment: fixed;
```

补充，这几个分别是：设置颜色，设置背景图片，设置背景图像距离元素左上角的水平和垂直偏移量；

图片是否重复（repeat-x 表示：**在水平方向上重复**背景图，垂直方向不重复。

其他常见值还有：

- repeat：水平和垂直都重复（默认值）
- no-repeat：不重复

- `repeat-y`：只在垂直方向上重复）；

图片是否随着页面滚动（`fixed` 表示：背景图**固定在视口**（屏幕）上，页面滚动时，背景图不会动

其他常见值：

- `scroll`：默认行为，背景图随着内容滚动
- `local`：背景图随着包含它的元素滚动（较少用））。

注释(和c差不多哎QAQ)

```
/* 这是一个注释 */
```

类型选择器（Type Selectors）

什么是选择器（selectors）？

它是一组用于匹配 HTML 元素的模式，用来告诉浏览器哪些元素应该应用接下来的样式。被选中的元素称为该选择器的“目标”。

之前就见过两种，分别是：

类型选择器（Type Selectors）

```
span {
  background-color: yellow;
}
strong {
  color: rebeccapurple;
}
em {
  color: rebeccapurple;
}
```

类选择器（Class Selectors）

```
.highlight {
  background-color: yellow;
}
```

还可以指定某种元素加某个类的组合:

```
span.highlight {
  background-color: yellow;
}
h1.highlight {
  background-color: pink;
}
```

这样样式就只应用于 `<span class="highlight">` 和 `<h1 class="highlight">`，而不是所有带有该类的元素。

你也可以选中拥有多个类的元素，比如：

```
.notebox.danger {
  border-color: red;
  font-weight: bold;
}
```

这个样式只会应用于同时拥有 notebox 和 danger 类的元素。

ID 选择器（ID Selectors）

ID 选择器以 # 开头。例如 #heading 只会选中 ID 为 heading 的元素

```
#one {
  background-color: yellow;
}
h1#heading {
  color: rebeccapurple;
}
```

注意：一个文档中 ID 只能使用一次，重复使用虽然有时能生效，但会造成无效的 HTML 和奇怪的行为。

选择器列表（Selector Lists）

多个选择器可以通过逗号（,）组合在一起，共享同一个样式：

```
h1, .special {
  color: blue;
}
```

补充：如果一个组合选择器中有任何部分是语法错误的，整个规则都会被忽略。

通配选择器（Universal Selector）

通配选择器用星号 * 表示，会选中所有元素。例如：

```css
* {
  margin: 0;
}
```

这个规则可以移除页面中所有元素的默认外边距。通常用于“重置样式表”，清除浏览器默认样式。

还可以用它提高选择器的可读性。例如：

```css
article *:first-child {
  font-weight: bold;
}
```

表示选中 `<article>` 元素内部任意第一个子元素。

属性存在与属性值选择器（Presence and value selectors）

这些选择器允许你根据某个属性是否存在，或属性值是否匹配，来选中元素。

选择器	示例	描述
`[attr]`	`a[title]`	匹配具有 `title` 属性的 `<a>` 元素。
`[attr=value]`	`a[href="https://example.com"]`	匹配 `href` 属性**正好等于** `https://example.com` 的 `<a>` 元素。
`[attr~=value]`	`p[class~="special"]`	匹配 `class` 属性中包含（以空格分隔的）`special` 值的元素。

选择器	示例	描述
[attr|=value]	div[lang|="zh"]	匹配与 attr 属性相匹配的元素，该属性的值要么与 value 完全相同，要么以 value 开头，后面紧跟连字符。

一个例子：

```html
<h1>属性存在与属性值选择器</h1>
<ul>
  <li>Item 1</li>
  <li class="a">Item 2</li>
  <li class="a b">Item 3</li>
  <li class="ab">Item 4</li>
</ul>
```

```css
body {
  font-family: sans-serif;
}

li[class] {
  font-size: 120%; /* 所有有 class 属性的 li 都加大字体 */
}

li[class="a"] {
  background-color: yellow; /* 仅匹配 class 精确为 "a" 的元素 */
}

li[class~="a"] {
  color: red; /* 匹配 class 含有单独 "a" 的元素，例如 "a b" */
}
```

子字符串匹配属性选择器（Substring matching selectors）

选择器	示例	描述
[attr^=value]	li[class^="box-"]	匹配属性值**以** box- 开头的元素。
[attr$=value]	li[class$="-box"]	匹配属性值**以** -box 结尾的元素。
[attr*=value]	li[class*="box"]	匹配属性值**中包含** box 的元素。

例子：

```html
<h1>属性子字符串匹配选择器</h1>
<ul>
  <li class="a">Item 1</li>
  <li class="ab">Item 2</li>
  <li class="bca">Item 3</li>
  <li class="bcabc">Item 4</li>
</ul>
```

```css
body {
  font-family: sans-serif;
}

li[class^="a"] {
  font-size: 120%; /* 匹配 class 以 "a" 开头的项（第1、2项） */
}

li[class$="a"] {
  background-color: yellow; /* 匹配 class 以 "a" 结尾的项（第1、3项） */
}

li[class*="a"] {
  color: red; /* 匹配 class 中任何位置包含 "a" 的项（全部4项） */
}
```

伪类(Pseudo-classes)

伪类是一种用于选择处于特定状态的元素的选择器，比如某元素是其类型中的第一个，或当前鼠标悬停在该元素上。伪类的行为类似于你在 HTML 中给某部分添加了一个类，它可以帮助你减少标记中的类名，使代码更加灵活和易维护。

伪类是以冒号开头的关键字，例如 `:hover` 就是一个伪类。

例如我们可以使用 `:first-child` 伪类来代替添加类名，它总是会选择文章中的第一个子元素，这样即使内容变动也不需要修改 HTML 结构。

```
article p:first-child {
  font-size: 120%;
  font-weight: bold;
}
```

常见的伪类还有：

- `:last-child`：选择其父元素中最后一个子元素
- `:only-child`：选择是其父元素中唯一子元素的元素
- `:invalid`：选择值不合法的表单字段

注意：你可以单独使用伪类，不一定需要加上元素选择器。例如，`:first-child` 会应用到所有作为其父元素第一个子元素的元素上，相当于 `*:first-child`。(不过通常为了精确控制，你还是应该加上元素名称。)

用户行为伪类User-action pseudo classes

有些伪类仅在用户与页面交互时才会生效，我们称这些为用户行为伪类或动态伪类。它们的作用类似于用户操作时自动给元素加上一个类名，例如：

- `:hover`：鼠标悬停在元素上时触发

- `:focus`：元素获得焦点时触发，比如点击或用键盘导航时

伪元素pseudo-element

伪元素的行为与伪类相似，但它们仿佛是往 HTML 中添加了新的元素，而不是给已有元素加类。

伪元素以两个冒号 `::` 开头，例如 `::before`。

（早期伪元素用的是单冒号，如 `:before`。现代浏览器对单冒号和双冒号都支持，但推荐使用双冒号写法。）

举个例子，如果你想选中段落的第一行文字，可以用 `::first-line` 伪元素。由于我们无法预知哪一段会换行，用 `<span>` 包裹第一行是不可行的，而 `::first-line` 会根据当前的文字长度、窗口宽度等自动只选中“第一行”。

```
article p::first-line {
  font-size: 120%;
  font-weight: bold;
}
```

使用 ::before 和 ::after 生成内容

有两个特殊的伪元素可以配合 `content` 属性使用，来向页面中插入内容，比如：

```
.box::before {
  content: "这段文字会出现在原始内容前面。";
}
```

```
<p class="box">这是页面中已有的内容。</p>
```

如果你将 `::before` 改成 `::after`，文字会插入在元素末尾。

(然而，通过 CSS 插入文字并不常见，因为这些文字对一些屏幕阅读器不友好，维护起来也不方便。一般就插入图标（如小箭头）。)

伪元素也常用于插入空字符串，这样我们可以像操作普通元素一样设置样式：

```
.box::before {
  content: "";
  display: block;
  width: 100px;
  height: 100px;
  background-color: rebeccapurple;
  border: 1px solid black;
}
```

这样就可以手动造图形了！

Descendant Combinator（后代组合器）

符号： 空格 " "

作用： 选中**后代元素**（不限层级，只要是祖先-后代关系）

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

```
<div class="box"><p>Text in .box</p></div>
<p>Text not in .box</p>
```

只有在 `.box` 里的 `<p>` 会被选中。

Child Combinator（子组合器）

符号： `>`

作用： 只选中**直接子元素**

Next-sibling Combinator（相邻兄弟组合器）

符号： `+`

作用： 选中**紧跟在某个元素后**的**下一个兄弟元素**

```css
h1 + p {
  background-color: #333;
  color: #fff;
  font-weight: bold;
}
```

```html
<article>
  <h1>A heading</h1>
  <p>第一段</p>
  <p>第二段</p>
</article>
```

注意，如果在 `<h1>` 和 `<p>` 中间插入其他元素，比如 `<h2>`，那么这个 `<p>` 就不会被选中。

Subsequent-sibling Combinator（后续兄弟组合器）

符号： `~`

作用： 选中**同级的所有后续兄弟元素**（不限是否是紧邻的）

```
h1 ~ p {
  background-color: #333;
  color: #fff;
  font-weight: bold;
}
```

则上例中两个 `<p>` 都会被选中，因为它们都出现在 `<h1>` 之后。

盒模型（Box Model）

在 CSS 中，页面上的所有内容都被看作是盒子（boxes）。

块级盒子与行内盒子

CSS 中的盒子主要分为两类：**块级盒子（block boxes）*和*行内盒子（inline boxes）**。每个元素都有一个**外部显示类型（outer display type）*和一个*内部显示类型（inner display type）**。一般我们使用 `display` 属性来设置盒子的显示行为。

`display: block`（块级盒子）的特征：

- 盒子会在新的一行开始；
- 可以设置 `width` 和 `height`；
- `padding`、`margin` 和 `border` 会将其他元素推开；
- 如果没有设置宽度，则默认会占据其父容器的所有宽度；
- 一些 HTML 元素默认是块级元素，如：`<h1>`、`<p>`。

`display: inline`（行内盒子）的特征：

- 盒子不会换行；
- `width` 和 `height` 设置无效；

- 上下方向的 padding、margin、border 不会影响其他行内元素的位置；
- 左右方向的 padding、margin、border 会推开其他行内盒子；
- 一些 HTML 元素默认是行内元素，如：<a>、、、。

CSS 盒模型

CSS 盒模型定义了一个盒子的结构，包括以下几个部分：

1. **内容区域（content box）**：显示文本或图像的区域；
2. **内边距（padding box）**：内容周围的空白；
3. **边框（border box）**：包裹内容和内边距的线框；
4. **外边距（margin box）**：盒子最外层的间距，用于与其他元素保持距离。

标准盒模型

在默认的标准盒模型中，width 和 height 属性只控制内容区域（content box）的大小，padding 和 border 会额外增加盒子的实际尺寸。

```
.box {
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```

$$\text{实际宽度} = 350 + 25 * 2 + 5 * 2 = 410px$$
$$\text{实际高度} = 150 + 25 * 2 + 5 * 2 = 210px$$

注：外边距不会算在盒子的宽高内，但它确实会占页面的空间。

替代盒模型（box-sizing: border-box）

在这个模型中，width 和 height 表示**整个盒子**的尺寸，padding 和 border 被包含在里面，计算更直观。

```
.box {
  box-sizing: border-box;
}
```

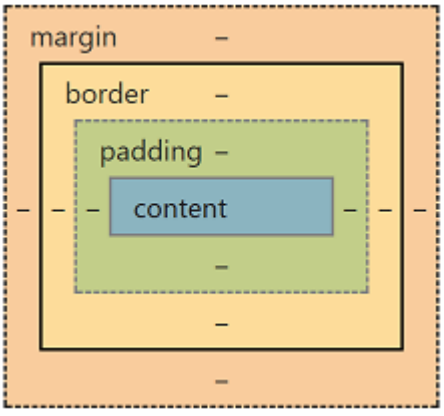
```
width: 350px;
padding: 25px;
border: 5px solid black;
```

意味着内容区域的宽度会是：

$350 - 25 * 2 - 5 * 2 = 290px$

(一般好像都用这个)

外边距、内边距与边框



外边距（margin）

- 是盒子外部的空白；
- 可设为负值，用来产生重叠；

- 可以使用简写属性或以下单独属性：
  - `margin-top`
  - `margin-right`
  - `margin-bottom`
  - `margin-left`

外边距合并（Margin Collapsing）

当两个垂直方向的外边距相遇时，它们可能会发生合并：

- 两个正值 → 取最大者；
- 两个负值 → 取最小者（数值更负）；
- 一个正一个负 → 值相加。

```
.one {
  margin-bottom: 50px;
}
.two {
  margin-top: 30px;
}
```

实际间距为 **50px** 而不是 80px。

如果 `.two` 的 `margin-top` 改为 -10px，最终间距变为 40px（50 - 10）。

边框（border）

- 在 `padding` 和 `margin` 之间；
- 在标准模型中，边框增加盒子的尺寸；
- 在替代模型中，边框从已有的宽高中“占用”部分空间；

- 可设置颜色、宽度、样式等。

内边距（Padding）

内边距是**内容与边框之间的空间**，它不会有颜色（除非背景色延伸），但可以扩大内容显示的“安全区”。

……总感觉这两部分没必要现在就开始记，等到要用再查也可以，遂skip。

处理冲突（Handling conflicts）

理解层叠机制（Understanding the cascade）

当多个规则应用于同一个元素的同一个属性时，CSS 通过以下三个因素（按优先级递增）来决定最终使用哪条规则：

1. **源顺序（Source order）**：相同特异性下，靠后的规则生效；
2. **特异性（Specificity）**：更具体的选择器优先生效（就是说优先级啦QAQ）；
3. **重要性（Importance）**：使用 `!important` 标记的规则会覆盖其他所有。（可以直接放在后面，但是慎用！）

不同情况下，优先级：

> 元素选择器（如 `h1`）特异性较低；
>
> 类选择器（如 `.main-heading`）特异性更高；
>
> 属性选择器、伪类选择器（如 `:hover`）的特异性与类选择器相同；
>
> ID 选择器的特异性最高。

控制继承的五个特殊值

CSS 提供了五个通用属性值，可以控制是否继承。它们适用于**所有属性**：

- `inherit`: 强制继承父元素的值；
- `initial`: 使用该属性的默认初始值；
- `revert`: 重置为浏览器默认样式（而不是属性初始值）；
- `revert-layer`: 回退到前一个层叠层中定义的值；
- `unset`: 如果该属性默认是可继承的，则等同于 `inherit`，否则等同于 `initial`。

```html
<ul>
  <li>默认链接 <a href="#">颜色</a></li>
  <li class="my-class-1">继承 <a href="#">颜色</a></li>
  <li class="my-class-2">重置为 <a href="#">初始值</a></li>
  <li class="my-class-3">取消设置 <a href="#">颜色</a></li>
</ul>

```

```css
body {
  color: green;
}

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

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

.my-class-3 a {
  color: unset;
}
```

重置所有属性（Resetting all properties）

`all` 是一个 CSS 简写属性，可一次性为几乎所有属性设置如 `inherit`、`initial`、`unset` 等值，适用于恢复到一个已知的初始状态：

设置大小

- `min-width` / `min-height`：设置元素的最小宽度/高度
- `max-width` / `max-height`：设置最大宽度/高度

常见用法：
为了让图片在父容器不足以容纳原始宽度时自动缩小，可以使用 `max-width: 100%`。
这样能防止图片被放大（防止像素模糊），但允许在必要时缩小以适应布局。

关于这部分，直接看https://xn4zlkzg4p.feishu.cn/wiki/KOebw5kNaiOLYtkWeUkcjJWgnNe吧！

视口单位（viewport units）

视口是你在浏览器中看到的网页区域。CSS 提供了基于视口大小的单位：

- `vw`：视口宽度的 1%
- `vh`：视口高度的 1%

```
.box {
  width: 20vw;
  height: 20vh;
  font-size: 10vh;
}
```

随着视口尺寸的变化，这些值也会实时变化。你可以用这种方式实现像“全屏展示”那样的效果，例如设置某一部分高度为 `100vh`，就可以撑满整个视口。

---

overflow 属性

overflow 属性可以帮助你控制一个元素内容的溢出情况。通过这个属性，可以告诉浏览器如何处理溢出的内容。overflow 的默认值是 visible，这意味着默认情况下，溢出内容是可见的。

如果你想在内容溢出时隐藏它，可以设置 overflow: hidden。这会把溢出部分隐藏起来。

如果你希望当内容溢出时**出现滚动条**，可以使用 overflow: scroll。此时浏览器会**始终显示滚动条**——即使内容没有溢出也会显示。这种方式可以保持布局一致，而不是因为内容变化导致滚动条忽隐忽现。

如果其实我们只需要在 y 轴滚动，但却出现了两个方向的滚动条。为了只在 y 轴上滚动，可以使用 overflow-y: scroll。

如果你只希望在内容溢出时才显示滚动条，使用 overflow: auto。浏览器会根据是否溢出来决定是否显示滚动条。

图片过大，超过盒子？用**object-fit**！

- cover：保持宽高比，缩放图片填满容器，　　但可能会裁剪部分内容；
- contain：保持宽高比，缩放图片以完全容纳于容器中，但可能出现“信箱”效果（留白）；
- fill：不保持宽高比，强制填满容器，可能导致图片变形。

字体继承问题

有些浏览器默认不让表单元素继承父元素字体。你可以使用如下 CSS 强制继承：

```
button,
input,
select,
textarea {
  font-family: inherit;
  font-size: 100%;
}
```

不同浏览器对表单元素的 box-sizing 使用规则不同，建议统一设置：

```
button,
input,
select,
textarea {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}
```

一些旧浏览器在没有必要时也会显示滚动条，因此你可以这样处理：

```
textarea {
  overflow: auto;
}
```

综上，CSS起步模板就是！

```
button,
input,
select,
textarea {
  font-family: inherit;
  font-size: 100%;
  box-sizing: border-box;
  padding: 0;
```

```
  margin: 0;
}

textarea {
  overflow: auto;
}
```

补充:

flex

grid