|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | CSS Selectors |  | |  |  | |
| This section is to introduce how to use CSS2 & CSS3 selectors |

# 30 ways to declare CSS Selectors

## **\* (All-Selector)**

\*{

margin:0px;

padding:0px;

}

首先我们来认识一些简单的选择器, 然后再介绍其它高级的选择器.

星号可以用来定义页面的所有元素. 许多开发者会使用星号选择器这个技巧来把margin和padding都设为0.

在快速开发和测试中这种设置固然是简单快捷的, 但并不建议在最终的产品代码中使用, 因为会给浏览器增加不必要的负荷.

## **#id (Id-Selector)**

## #selector {

## width: 960px;

## }

## 这是CSS选择器中的id选择器.

## ID选择器作为前缀可以选择该id的元素. 这是最常见的用法, 但使用id选择器时需要谨慎, 因为id选择器比较局限, 不能重用. 如果可以的话, 先尝试使用标签名称或者HTML5的其中一个新元素或者是实用伪类.

## .**class (Class-Selector)**

## .selector{

## color: red;

## }

## 这是CSS选择器中的class类选择器. Id和class类选择器的区别是, 类选择器可以定义多个元素.

## 当我们想定义一组元素的样式的时候可以使用class选择器.

## **selector selector (Descendant-Selector)**

## li a {

## color: red;

## }

## 这是CSS选择器中的后代选择器, 当你需要更精确的定位的时候可以使用后代选择器.

## 例如说, 当我们只想选择无序列表里的链接, 而不是所有链接, 我们就可以使用后代选择器. 但千万注意不要使用多重后代选择器, 这样会严重影响CSS的性能和负荷.

## **X (Label-Selector)**

## a {

## color: red;

## margin: 30px;

## }

## 这是CSS选择器中的标签选择器.

## 如果我们想让定义的页面中所有相同标签的类型都应用同一种样式, 我们就可以选择标签选择器.

## **X: visited & X: link (Pseudo Class-Selector)**

## a: link { a: visited {

## color: red; color: blue;

## } }

## 这是CSS选择器中的[链接]和[已访问]伪类选择器.

## 我们可以使用 : link 伪类来定义所有还没有点击过的链接.

## 我们可以使用 : visited 伪类来定义所有曾经点击或者访问过的链接.

## **X + Y (Nearest-Selector)**

## ul + p {

## color: red;

## }

## <ul> … </ul>

## <p>

## <p>

## 这是CSS选择器中的最近元素选择器.

## 所谓最近元素选择器, 是指只会选中紧挨在定义元素后面的匹配元素. 在上面的例子中, 这一段样式只会影响在ul后面的第一个段落p.

## **X > Y**

## div#container > ul {

## border: 1px red solid;

## }

## 这是CSS选择器中的直属元素选择器.

## 所谓直属元素选择器, 是指在定义元素内的直属子元素的匹配元素.

## 如果, 我们按照上面定义的样式, 应用到下面的页面样式时:

## <div id=’container’>

## <ul>

## <li> List Item

## <ul>

## <li> Child Item </li>

## </ul>

## </li>

## <li> List Item </li>

## <li> List Item </li>

## </ul>

## </div>

## #container > ul 只会影响定义在id为container的div里面的第一层(直属关系)ul, 而不会影响在li中的ul样式.

## 因此, 这种选择器的定义更为明确, 这也在JavaScript中可以快速应用.

## **X ～Y**

## ul ~ p {

## color: red;

## }

## 这是CSS选择器中的兄弟选择器, 它和X + Y 选择器很像, 但是影响范围完全不同.

## 邻近关系选择器只会影响紧挨在定义元素后面的第一个匹配元素, 但是兄弟选择器则会影响紧挨在定义元素后面的所有匹配元素而并非第一个.

## 重要的是, 兄弟选择器的页面结构如果被打断, 则影响范围将被打断.

## **X[title]**

## a[title] {

## color: red;

## }

## 这是CSS选择器中的属性选择器. 后面会介绍更多的属性选择器的使用方法.

## 在上面的样式定义影响下, 所有带有title属性的a标签才会被匹配和影响, 不带有title属性的a标签将不会被影响.

## **X[href = ”URL”]**

## a[href = “http://www.google.com”] {

## color: red;

## }

## 这是CSS选择器中的属性选择器, 但是特定的目标是所有指向 http://www.google.com 的链接, 它们的字体将会是红色.

## 请注意我把地址放在引号中, 在JavaScript 中选择元素的时候也要用这种方式使用, 尽可能的按照CSS3的标准来使用选择器.

## 但是这样的选择器也有它的局限性, 如果我们的地址并非使用绝对路径而是使用相对路径, 那么我们的选择器将不会生效.

## 要避免此类情况发生, 我们可以使用正则表达式来代替.

## **X[href\* = “URL”]**

## a[href\* = “google.com”] {

## color: red;

## }

## 我们进一步的给地址属性选择器加上通配符 \* (星号), 这样就能解决了上面描述的问题.

## 星号表示赋值可以出现在属性的任意处. 这样的定义可以指向 www.google.com 同样也可以指向 gmail.google.com

## 这种选择器的指向范围仍然很宽广, 也可能会影响到我们在页面上自己应用的google 产品链接.

## 如果我们想更精确的缩小影响范围, 我们可以使用 ^和$来匹配前缀和后缀.

## **X[href^ = “URL”]**

## a[href^=”http”] {

## background: url(‘path/externalURL/icon.png’);

## color: red;

## }

## 这是CSS选择器中的前缀匹配选择器, 所不同的是在这个例子中结合了地址选择器.

## 如果我们在项目中需要给所有指向外部链接的超链接加上特定的图标, 那么我们就可以使用这样的带有正则表达的特定属性选择器.

## 需要注意的是 ^ 是用来指向特定的前缀开头, 我们用 ^=’http’ 这样的表达式, 它既可以影响所有以 http:// 为开头的超链接, 还能影响 https://

## **X[href$ = ‘.jpg’]**

## a[href$=’.jpg’] {

## color: red;

## }

## 这是CSS选择器中的后缀匹配选择器, 所不同的是我们这这里结合了地址选择器.

## 我们使用$正则来声明我们所期望匹配的后缀, 在这里我们匹配的是所有以 .jpg 为后缀的超链接.

## 它的缺点就是我们无法指向 png 或者 gif 图片. 我们当然可以用更为特定的属性选择器来指定图片类型.

## **X[data-\* = ‘image’]**

## a[data-filetype = ‘image’] {

## color: red;

## }

## 这是CSS选择器中的自定义属性匹配选择器. 这是方便我们将一种统一的模式去遍历和影响所有符合条件的元素.

## 按照上面的表达式, 我们定义了 data-filetype = ‘image’ 的元素. 那么我们在页面上添加任何图片类型的标签时, 只要按照这样的统一规范:

## <a href=’path/profile.jpg’ data-filetype=’image standard internal’> Image Link </a>

## <a href=’path/user.png’ data-filetype=’image external’> Image Link </a>

## 无论href的后缀为.jpeg, png, bmp或者gif, 我们都可以用这种统一规范的选择器对其进行样式的调整.

## **X[Y ~= ‘Z’]**

## a[data-info~=’external’] {

## color: red;

## }

## a[data-info~=’image’] {

## border: 1px red solid;

## }

## 这又是一种正则表达式的选择器, 波浪符号(~)的作用是可以让你定义取值带有空格的属性.

## 继续使用前面自定义的属性表达式, 这次我们创建data-info这个属性来匹配带有空格取值. 举个例子:

## <a href=’path/profile.jpg’ data-info=’external image’> External Image </a>

## 按照我们定义的两个CSS样式, 通过(~)通配符, 我们找到了页面上的这个符合条件的标签.

## 因为这个标签符合这两个条件, data-info中, 既包含 external 又包含 image 所以它将被这两个CSS样式重定义它的样式.

## 如果页面上存在另一个元素满足其中一个条件, 那将会被满足条件的CSS样式所影响.

## **X:checked**

## input[type=radio]:checked {

## border: 1px red solid;

## }

## 这是CSS选择器中的伪类选择器 checked. :checked属性经常被应用在 input[type=checkbox] 和 input[type=radio] 这两个页面元素上.

## **X:before 和 X:after**

## li:before { li:after {

## content: ‘’; content: ‘’;

## clear: both; border: 1px gray solid;

## } }

## 这是另外两个CSS选择器中的伪类选择器, before和after被应用于在元素前和后添加内容.

## 常见的案例是我们在menu中经常用before和after来创建小箭头而并非用图片, 因为图片会因为分辨率的不同而失真.

## 我们也常用这两个伪类在元素后面添加间隔和清除浮动的元素. 根据标准的CSS3写法, 伪类声明前应该使用双冒号, 然而为了兼容更多的浏览器, 单冒号也被承认.

## 所以在实际使用情况中, 单冒号是被广泛使用并认可的.

## **X:hover**

## div:hover{

## background: red;

## }

## 这是另一个常被应用的伪类选择器, 作用是当鼠标悬停时候产生自定义的样式变化.

## 我们可以把这些页面元素经常被广泛应用的伪类称之为[用户交互伪类]

## **X:not(selector)**

## div:not(#container) {

## color: blue;

## }

## 这是CSS伪类选择器中的另一个灵活应用的伪类, 我们称之为 否定伪类选择器.

## 如果页面元素中包含了N个div, 但是我们想让非特定的div变成我们希望的样式, 那么我们就可以使用否定伪类选择器.

## 这是一种排除法的技巧, 当页面元素是动态生成或者说是非稳定结构的时候, 它会变的非常有效.

## **X::first-line 和 X::first-letter**

## 这是CSS伪类中常用的段落伪类, 我们用这种伪类来定义某些片段的元素, 这里我们定义了第一行和首字母.

## 很重要的一点是, 这样的伪类选择器只能用在块状元素 (block element)

## **X:nth-child(n)**

## li:nth-child(3) {

## color: red;

## }

## 这个CSS伪类是为了可以匹配一组序列元素里的某个特定元素. 我们称之为 特定元素选择器.

## 但要注意的是, 它和JavaScript中的索引值不同, 它的序列是从1开始而不是以0为起点. 例如我们想选择li中的第二个元素, li:nth-child(2)

## 这里的参数还有很广泛的用途. 比方说我们想将样式应用在一系列的子元素上. 比如说 li:nth-child(4n) 可以选择排在4倍数上的元素.

## li:nth-child(even) 和 li:nth-child(odd) 则是用两个关键字even和odd 来指定奇偶行元素.

## **X:nth-last-child(n)**

## li:nth-last-child(3) {

## color: red;

## }

## 这个伪类选择器是 :nth-child的倒序方式. 如果nth-child 是指从头开始, 那么 nth-last-child 就是指从尾开始.

## 我们经常会遇到无序的元素集合, 如果我们必须在特定的情况下给这个无序的集合中倒数第三个元素指定样式, 我们就可以用这个伪类选择器.

## **X:nth-of-type(n)**

## div:nth-of-type(3) {

## color: red;

## }

## 这是正序类型匹配选择器, 和 nth-child(n)类似, 只是它匹配的不是元素而是元素类型. 如果我们页面上存在N个无序div元素, 但是我们想将第三个div 改变样式, 我们就可以用正序类型选择器进行定位.

## **X:nth-last-of-type(n)**

## div:nth-last-of-type(3){

## color: red:

## }

## 这是倒序类型匹配选择器, 和nth-last-child(n)类似, 匹配的是类型, 作用是倒序的方式进行定位.

## **X:first-child**

## li:first-child {

## border-bottom: none;

## }

## 这个首元素伪类选择器也是我们在CSS中常用的选择器, 我们经常在Menu或者Navigation中遇到这样的样式定义, 作用经常是去掉边框或者是首元素特定样式的定义.

## **X:last-child**

## li:last-child {

## border-top: none;

## }

## 与 :first-child 相对的, 就是尾元素伪类选择器. 相同的使用方式, 在 :first-child 定义的地方也常常会声明 :last-child 的样式

## **X:only-child**

## ul > li:only-child {

## color: red;

## }

## 仅有元素选择器, 常常被用作于高亮样式. 意思是当匹配的唯一子元素这个条件存在时, 我们对其进行样式的定位.

## 比方说我们得到两个数据列表, 我们将内容以 ul 和 li 的形式进行页面渲染.

## <ul> <ul>

## <li> Only child </li> <li> List 1 </li>

## </ul> <li> List 2 </li>

## </ul>

## 当出现仅有元素匹配的情况下, 样式会被应用. 这两个ul中, 只有第一个ul中的li会被样式影响, 因为它仅有一个子元素 li

## 这常常被我们用于仅有数据的高亮和警告.

## **X:only-of-type**

## div > span:only-of-type {

## color: red

## }

## 仅有类型选择器, 我们需要将它与 :only-child 区分. 虽然它们的作用类似, 但是意义不同. 例如:

## <div> <div>

## <p> One paragraph </p> <span> Only one Span </span>

## <span> One span </span> </div>

## </div>

## 我们需要考虑一下, 如果存在这样两个页面元素, 到底哪个span元素会被样式重写.

## 其实它们(这两个span)都会被样式影响. 因为 only-of-type 只会判别在所有子元素类型中是否仅有被定义的唯一的特定类型, 如果存在就将被样式影响.

## **X:first-of-type**

## div > p:first-of-type {

## color: red;

## }

## 首匹配元素类型选择器, 简单明了的定义. 无论在div中定义了多少个p元素, 只要匹配 div父结构中包含的第一个出现的p子元素, 它就会被样式影响.

# Summary

## 值得重视的是, 虽然我们列举了如此纷繁复杂的CSS选择器, 但是我们在实际项目中需要尽可能避免的去使用多重复杂的条件式选择器, 也尽量减少集合判断式的选择器.

## 因为它们会严重影响CSS样式的运行效率. 另外, 我们在开发中会无意识的在页面元素上添加 style 然后定义它们的样式. 这会给CSS带来另一个严重性的问题: 样式的优先级

## 样式的优先级也是给我们造成困扰的因素, 这往往与我们的开发习惯有关, 但更重要的是我们必须清楚在CSS中样式的优先级是如何判断和最重渲染的.

## 我们定义样式的方式大致有下面几种:

## Inline Style (普通且常见的写法, 写在HTML元素的style中) <div class=’abc: color:blue’ style=’color: red’>

## Embedded Style (也很普遍, 在HTML的header中我们声明了<style>标签, 然后在里面写了CSS样式)

## External Style (项目最后开发中, 我们将样式整合在外部的 .css 文件中, 以 <link> 的方式将其引入)

## 其中最常见的样式重写冲突发生在条件1和条件3中:

## !important 带有重要性声明的样式优先级最高

## 其次是 inline-style 的优先级, 行样式声明高于外部声明, 外部声明是指通过 <link> 和 <style> 来声明的CSS样式

## #id 选择器的优先级高于 .class 选择器 (简单的描述就是, 越简明精确的定位方式, 它的优先级越高)

## 如果出现同类型的选择器定义的样式, 则后出现的样式覆盖前出现的样式

## 这些是我们常见的CSS冲突中的典型案例. CSS的优先级其实是有它独特的算法而计算出来的.