# Performance & Organization [性能和组织]

对编写HTML和CSS有扎实的理解是一项非常出众的专业技能，随着网站代码量和流量的增加，一项在开发时间和用户体验上都很重要的技术组合开始流行。明白网站的性能和组织结构的原理是长期有效的方法。

基础代码的组织和架构不仅影响开发速度而且对页面展示速度也有很大影响。这两者均值得开发者和用户强烈关注。花时间为基础代码设计合理的结构，并且标识出所有不同的组件是怎么合在一起工作的，这样可以加快产能并且可以了解各个方面（的作用）。

另外，对网站性能做出一小点改变可以得到很大成效。网站性能更类似于二八定律，意为20%的优化可以大约让网站提速80%。

## --策略和结构--

提高网页性能和架构的第一部分就是围绕开发基础代码定义出优秀的策略和架构.特别要指出的，要建立强而有力的架构，列举出设计模式并且找出可以重用常用代码的方法。

### 1）style 架构

准确来说使用怎样的样式结构取决于个人喜好和给定的页面怎样是最好的，但是通常来说会有最好的实践值得去使用。有这样的一个基于目的划分style的实践，它创建的目录包括通用基础样式，用户接口组件以及商业逻辑模型。（目录结构如下）

# Base

– normalize.css

– layout.css

– typography.css

# Modules

– aside.css

– footer.css

– header.css

# Components

– alerts.css

– buttons.css

– forms.css

– list.css

– nav.css

– tables.css

上面这种架构列出来的包括三个目录，每个都是独立的样式组。这里的目的是让大家开始吧网站当做一个系统来考虑而不是一个个独立的页面，上述这个代码结构就表达了这种思想。你可以注意到这套架构中没有任何特定页面的样式。

在“Base”目录中以layout和typography为例里面包含了通用的样式和可用于整个网站的一些变量；在“components”目录中包含了根据特定用户接口元素来划分成诸如alerts和buttons组件文件的样式；最后一项，在“modules”目录中包含了页面根据商业需要而划分的各分块的样式。

“components”的样式只是单纯的接口需要，和网站核心业务逻辑是没有关联的。"modules"包含的样式则是针对特定业务逻辑的。当创建一个HTML模板是通常会用到不同的用户接口组件。例如，页面的侧边栏（sidebar）也许list和button样式是在“components”中定义的而其余部分的样式又是从“module”中继承的。这样的样式划分既鼓励在开发之前考虑清楚同时也能让样式有更广的共享性和重用性。

这种样式管理的策略并不是新近出现的，它在先前出现的OOCSS（Object Oriented CSS，面向对象的css）和SMACSS（Scalable and Modular Architecture for CSS，可扩展可模块结构化的CSS）的方法论中都有提及。这些方法论不仅在结构上而且在如何使用样式上都有自己的见解。

### 2）Object Oriented CSS（面向对象的CSS）

面向对象方法论是由Nicole Sullivan在她为大型网站编写样式时提出的。OOCSS定义了两个主要准则用于构建结构强健并代码量合理的可扩展网站。这两个准则包括：

--1--将结构从皮肤（样式）中分离【结构和样式分离 HTML和CSS】

--2--将内容从容器中分离

<要做到结构和样式分离甚至要把布局的元素从网站主题中抽离出来。> 模块的结构必须透明，允许其中的样式被继承并且可以无冲突的展示。最一般的例子是当某处需要“solid”的“grid”（网格）布局结构时，也能应用“crafted”（混合结构）的模板。

将内容从容器分离主要是移除父元素嵌套子元素的从属关系。一个“heading”部分即便没有包含在父容器中也要看起来是一样的，为了做到这一点，“heading”里的元素是有的默认样式然后在需要的时候从多个类中扩展

-------HTML-------

<div class="alert alert-error">

<p class="msg">...</p>

</div>

-------HTML-------

-------CSS--------

.alert {...}

.alert-error {...}

.msg {...}

-------CSS--------

OOCSS鼓励创建组件目录，并保持灵活和使用网格布局。这些是很好的基础规则并且它们可以帮助你避免每次往网站加页面或性能时都要添加额外的样式。

### 3)Scalable & Modular Architecture for CSS(可扩展并模块化架构的CSS)

和OOCSS有同样地位的方法论是由Jonathan Snook开发出来的。SMACSS提出把样式划分为五个主要的类别。它包括：

--Base

--Layout

--Module

--State

--Theme

"base"类别包括核心元素样式，囊括了大体默认的样式。"layout"类别中定义了不同元素尺寸和网格模式，用于决定它们的布局。"module"则是针对页面各独立部分更有针对性的样式，诸如导航栏或个性块的样式。"state"样式用于加强或者在模板包含警告状态时重写该事件样式，例如激活tab标签。最后，"theme"类别也许在皮肤，外观和感觉的基础上给不同模块添加样式。

-------HTML-------

<div class="alert is-error">

<p>...</p>

</div>

-------HTML-------

-------CSS--------

.alert {...}

.alert.is-error {...}

.alert p {...}

.alert.is-error p {...}

-------CSS--------

在上述这个例子里alert类是"module"类别中的而is-error类是"state"类别中的。这些类别中的样式在需要的时候可以继承。

### 4)选择一个方法论

无论选择使用哪一种方法论，完全取决于对要完成的网站的哪一个感觉更好。一般说来，一种OOCSS和SMACSS的可行的混合模式会比较好，可以从各种观点中借鉴你喜欢的。

## --性能驱动的选择器--

一个功能性强的CSS经常滥用但却没有意识到的一处就是选择器。对于CSS的关注一般是属性和值。所以当这些样式被应用到正确的元素上，一切看起来是那么美好。这是非常糟糕的想法。用CSS怎么选择元素是很影响性能的，不仅关系到页面加载同时对整个网站架构的样式实践和模块化有影响。

### --1)让选择器尽量短些

让CSS选择器名保持尽可能短有一些好处。可以最小化差异性，带来更好的继承和便携性，并且可以提高性能。长久以来，所用被认可的选择器减低了网站性能是因为他们强制浏览器从左往右渲染（加载）每个独立的选择器。他们同时也在其他选择器上加一些负载（标示）让其变得更加特别。

/\* Bad \*/

header nav ul li a {...}

/\* Good \*/

.primary-link {...}

/\* Bad \*/

button strong span {...}

button strong span .callout {...}

/\* Good \*/

button span {...}

button .callout {...}

上述代码中,第一个选择器非常特别但也可以这么定义也可以渲染到页面上，比单纯用类名要快些。但是，在这个例子中使用类名可以大大降低定义父元素的必要，用以保证这些元素位置在任何时间改变也不会破坏样式。

第二个例子包含的选择器比第一个例子的要短但是它可以通过对每一个选择符提供同一级别的特性。避免过度使用特性选择符，这样可以减少在元素结构变化时样式被破坏。剔除一些独立的选择器单元，给所有选择器一样的强度让它们更好操作。

缩短选择器名的一切目的是为了减少其差异性，以此创建更简洁更易扩充的代码。

### --2)偏爱classes

classes很好，它们渲染快速并且允许里面的样式被重用，并且也已经普遍用于构建网站了。然而当使用class的时候，有一些值得研究的实践用来验证是否使用得当。

既然选择器是从右往左渲染的那么关注核心选择符则显得尤为重要。核心选择符是在选择器末尾的单元，是最右边的。正因为核心选择符是浏览器要寻找定义的第一个元素，所以它显得尤为重要。核心选择符选的不好将得不到浏览器良好的支持。为了更好的性能不要害怕使用class定义更特别地样式。

还有，不要使用元素标签做类名的前缀。这样做的话一来会让该样式不那么容易被其他的元素标签应用并且把整个选择器的特异性都提高了。

/\* Bad \*/

#container header nav {...}

/\* Good \*/

.primary-nav {...}

/\* Bad \*/

article.feat-post {...}

/\* Good \*/

.feat-post {...}

尽可能远离ID选择符因为它们特异性很大并且不允许任何的重复。最后要说的是使用ID和使用!important并没有很大的区别。

## --可重用的代码--

性能缺陷绝大部分来自过于庞大的文件所带来的浏览器不必要的加载。尽可能的重用样式是帮助减小CSS文件尺寸快速有效的办法。任何重复的样式或接口模式都应该被整合起来用于代码共享。如果两个模板共享同样的背景，圆角和边影那就没有必要为两个模板都详述两次一样的样式了。用一个class就可以把他们结合在一起，使得它可以一次编写就被一直共享了。

重用代码在语义上也不会产生什么代价。有一种技术是让选择器成对出现并用逗号分隔，允许相同的样式可以通过两个选择符被继承。另一种可行的办法在之前提到的OOCSS和SMACSS方法论提到过，意为给一个class绑定样式后在同一个元素上使用多重classes。

/\* Bad \*/

.news {

background: #eee;

border-radius: 5px;

box-shadow: inset 0 1px 2px rgba(0, 0, 0, .25);

}

.social {

background: #eee;

border-radius: 5px;

box-shadow: inset 0 1px 2px rgba(0, 0, 0, .25);

}

/\* Good \*/

.news,

.social {

background: #eee;

border-radius: 5px;

box-shadow: inset 0 1px 2px rgba(0, 0, 0, .25);

}

/\* Even Better \*/

.modal {

background: #eee;

border-radius: 5px;

box-shadow: inset 0 1px 2px rgba(0, 0, 0, .25);

}

选择何种方法并没有很大区别，但代码被共享和重用确实可以大幅降低文件大小。

## --减小和压缩文件--

简单的移除重复和不必要的代码是降低文件尺寸的有效方法，但还有其他的途径。一类是减小和压缩诸如HTML,CSS和JavaScript文件。另外则是对图片进行压缩，移除图片的一些注释和介绍。

### --1)gzip 压缩

一种最流行的文件压缩方式叫做gzip。gzip会获取一般的包含HTML,CSS,JavaScript 在内的文件进而获取类似的字符串对文件进行压缩。在各文件中定义的匹配字符越多，文件可以被压缩的更小，然后就可以从服务器端把更小的文件发送到浏览器中。

设立gzip是完全不费事的，并且HTML5 Boilerplate里已经有一套很好的关于如何使用这些的方法。使用gzip 压缩文件需要在网站服务器的根目录里添加名为.htaccess的文件，并把一些特定文件标为压缩项。文件名前面的点是正确的，因为.htaccess文件是隐藏文件。

在HTML5 Boilerplate里提到在.htaccess文件中有些部分是用于gzip压缩的。在你自己服务器中重用.htaccess文件的全部或部分内容有利于建立gzip压缩。另外，如果单单是.htaccess文件在Apache web服务器上工作是没有用的，同时要以下的模板都启用：

mod\_setenvif.c mod\_headers.c

mod\_deflate.c mod\_filter.c

mod\_expires.c mod\_rewrite.c

一般来说没启用这些模板也不会有问题，一些服务器甚至也会为你开始做压缩处理。但不管怎么说，在服务器中压缩文件时最大的利益。

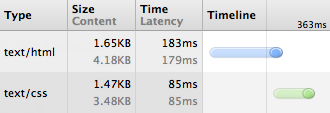
### --2)管理压缩

Google的chrome浏览器的网络探针可以提供一系列的性能数据，特别是网络

这块的数据。另外，有一些网站也可以帮助确认gzip压缩是否启用。

**Fig. 1** .01

该网站定义出了浏览器加载和显示每个文件的大小和时间。可以看到通过gzip压缩使文件减小了大约60%.



**Fig. 1** .02

观察浏览器对特别定义的文件支持何种压缩编码方式。在这个例子中，从request header中的记录可看到gzip，deflate和sdch都是被支持的。

从定义的response header的反馈可以看出该文件是使用了gzip编码方式进行压缩的。

### gzip Detail Screenshot --3) 图片压缩

压缩文件是有好处的，但通过压缩图片尺寸可以获得更好的效果。所有的图片文件量通过网站会很快累计起来，但是通过压缩图片可以很大程度上文件量至于可控范围。

很多人不选择对图片进行压缩是担心因此会减低图片本身的质量。大体来收这种想法是不对的并且图片是可以进行无损压缩，可以通过移除一些颜色配比和图片注释但不会影响图片质量。

这里有一些可以用于压缩图片的工具，最好用的两个是用于Mac的[ImageOptim](http://imageoptim.com/)和Windows的[PNGGauntlet](http://pnggauntlet.com/)。这两个服务都可以压缩大部分常用的图片格式，特别是JPG和PNG的图片文件。

值得一提的事，在HTML中通过width和height属性设置图片尺寸也可以帮助页面更快速的加载，会为图片边缘设置适当的空白。但要知道，这些属性只是用来定义确切的图片尺寸而不是用来缩小图片。使用的图片过大时，用width和height 把图片缩小是不好的做法，会导致更多不必要的加载。

## --减少HTTP请求--

相较于文件量，网站的HTTP请求数量也是是最大的性能痛点之一。每向服务器发送一个请求页面的加载时间就会增加。一个请求的开始必须要等另一个请求结束，太多的请求就会使服务器负载过大。

### 像文件一样合并

有一种方法，或许也是最简单的方法，就是像合并文件一样去减少HTTP的请求数量。特别要说的，把所有的CSS文件整合到一个CSS文件对JavaScript也是这样。合并这些文件然后把它们压缩就有希望得到少量的HTTP请求。

<!-- Bad -->

<link href="css/reset.css" rel="stylesheet">

<link href="css/base.css" rel="stylesheet">

<link href="css/site.css" rel="stylesheet">

<!-- Good -->

<link href="css/style.css" rel="stylesheet">

通常说来，CSS文件应该在文档（HTML）的head开头部分进行加载，而JavaScript应该放在页面底部加载，就放在</body>标签前面。这种特别地处理方式是因为CSS可以在页面边加载的时候也能跟着加载。而JavaScript，一次只能加载一个文件，这样会停止其他的任何加载。有一点要注意的事当JavaScript在做异步加载时页面本身也是在自我进行渲染的。另一个要特别注意的是JavaScript还要用于帮助渲染页面，就像HTML5shiv 这样的例子。

### 图片拼合

使用CSS的图片拼合操作包含了多个元素使用同一背景图片。目标是通过使用多种背景图片减少HTTP请求的数量。

创建一个拼合图片需要一些背景图片，一些常用的然后把它们整合到一张图片里。之后使用CSS把拼合的图片作为元素的背景图片，并使用background-position属性选出要正确显示的图片。

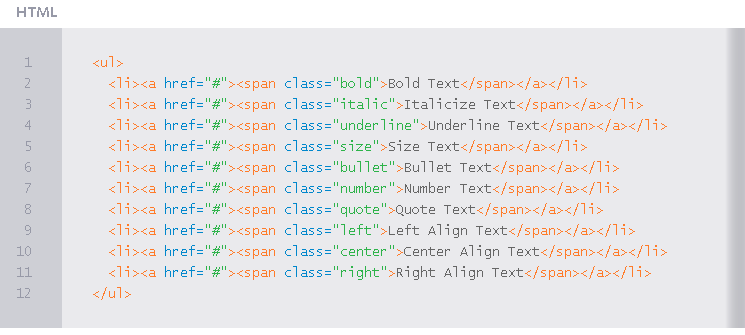
想象背景图片在元素背面滑动，仅在给定元素背后展示合适的背景。举例来说，如果一个元素是16px\*16px的那么也只能展示16px\*16px的背景图，拼合图片其他部分会在背后隐藏着。

**Fig. 1.03**

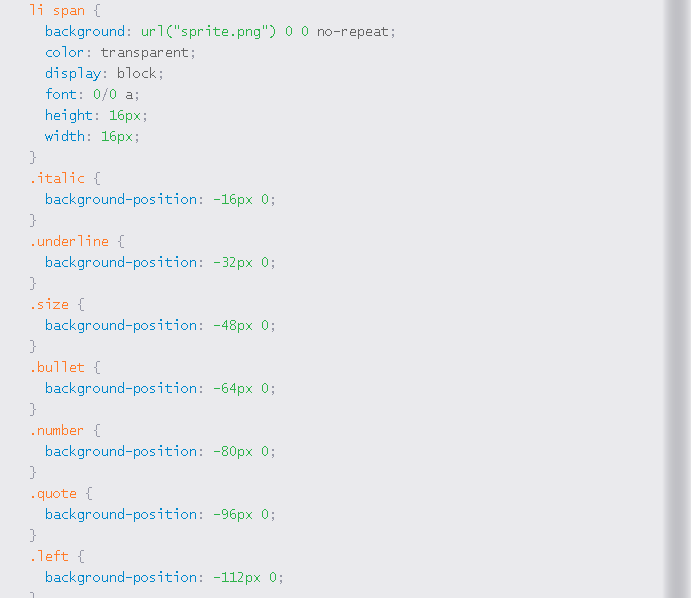
这里是一个文本编辑菜单的拼合图片，列出了图片背景位置会怎么改变的参考指南。

Menu Sprite

使用上述的拼合图片，可以在span标签上使用该拼合图片进行创建菜单。然后使用classes改变拼合图片的背景坐标，就可以实现不同图标的展示。



--CSS--





### 图片数据URI（Data-URI）

另外，相较于拼合图片，通过Data-URI可以把图片的编码数据直接用HTML和CSS包含进去，也避免了HTTP请求的必要。对小图片来说DATA-URI很不错，好像什么都没有变但是HTML和CSS可以得到高速加载。但是DATA-URI也有一系列问题，他们不容易修改和操作，导致需要进行其他的编码。并且他们在特别像IE7及其低版本中不能适用。

如果适用URIs可以帮助减少一些HTTP请求，并且HTML和CSS可以快速加载，那么好处是大于风险的。有一些可以帮助创建DATA-URI的工具：[converters](http://websemantics.co.uk/online_tools/image_to_data_uri_convertor/)和[pattern generators](http://www.patternify.com/). 但还是要注意，要确保实际的DATA-URI是比图片本身的尺寸要小的。

## --缓存通用文件—

另一个有效帮助减少HTTP请求，并且提高页面请求速度的方法就是缓存通用文件。当一个页面第一次加载时特定的文件才会被缓存。现在浏览器就不用再花同样的时间为相同的文件发出请求。多长的时间完全取决于你，取决于你希望用户为这个特定的文件类型等待多长时间。

对于gzipping过的文件，可以在.htaccess文件中为要缓存的文件设置过期的header。同理，在前面我们说到的一步中的HTML5 Bolierplate，里面的.htaccess文件也包含了一个expires headers的板块用来指定。

图片，视频，网页字体和常用的媒体类型经常设置为缓存一个月，而CSS和JavaScript通常是设置为一年的缓存。如果CSS或其他文件一年不止变化一次那需要为其更改名字，特别是版本，主要是为了加载需要。另外，expires headers可以在一段时间内做微小的改变。

ExpiresByType text/css "access plus 1 year"

ExpiresByType application/javascript "access plus 1 year"

把“access plus 1 year”改成“access plus 1 week”对于那些一周变一次并且没用使用文件名分隔来控制版本的CSS和JavaScript文件来说是更合适的。对于可以接受的expires headers值可以参照[syntax](http://httpd.apache.org/docs/current/mod/mod_expires.html)里的mod\_expire.

# 2. Detailed Positioning [详细定位]

当在页面建立布局和对内容进行定位的时候有很多不同可用的技术。选择使用何种技术很大程度上取决于内容和页面的目标，因为(基于这些因素）有些技术也许比其他的更好。

举个例子，对于有很多不同元素的页面边对边的浮动元素功能可以提供一个漂亮简洁的布局。然而，如果需要更加严格的控制，元素需要用到其他技术进行定位，包括relative和absolute定位。

这一课里，我们将开始学习怎么包含浮动。跟着学习，我们会涉及到很多定位技术，包括怎么精确一个元素的x和y甚至z坐标的定位。

## --包含浮动元素--

在创建网站布局时浮动元素是自然的过程，并且在页面中这是很常用的方法。浮动允许元素贴近或远离另一个元素。它们在基于自身的尺寸和父容器的尺寸下允许元素建立自然的流布局让元素进行交互。

当开始流动布局时，一个元素的位置是由它旁边的元素位置决定的。一个元素会跑进挨着它的那个元素里面吗？一个元素会出现在新的一行吗？这些全都取决于DOM（文档对象模型）和它周围的元素。

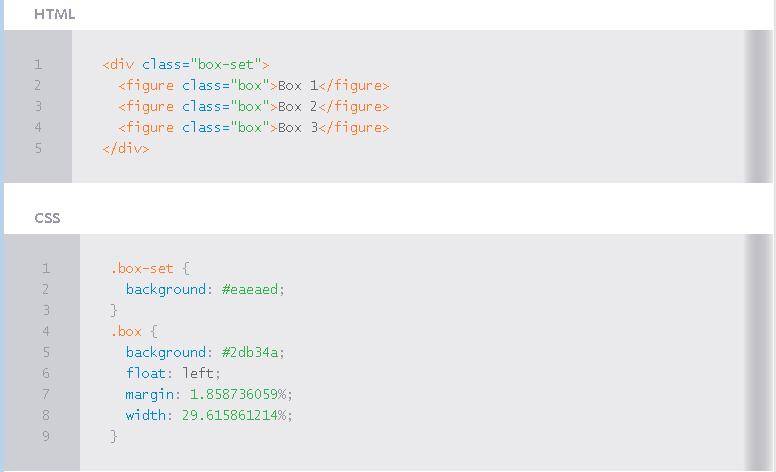
\*什么是DOM？

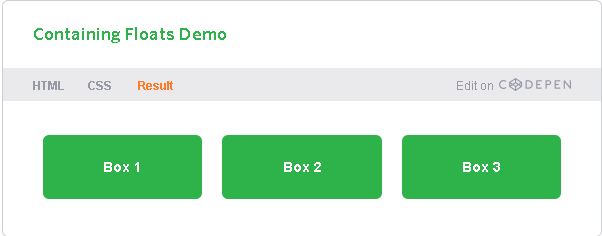
DOM，或者说文档对象模型，是一个给HTML和XML文档提供结构化描述的API。在我们的例子里，我们特指HTML文档，DOM描述了所有不同的元素和他们之间的关系。

这些描述可以被看着是一棵树的集合，每个元素和它周围的元素都有不同的关系。嵌套在别的元素里面表示有父子关系，而共有一个父级的元素存在兄弟关系。

浮动(Float)确实提供了非常强劲的力量，但他们也有自身的一些问题。最常见的问题是关于父元素包含一定数量的浮动元素。页面上的内容将会保留浮动子元素的尺寸和位置，但这些浮动的元素不会在意父容器的外部边缘。在这种情况下父元素就不知道在它内部具体包含了什么而导致崩塌，也即父元素的高度(height)会是0并且完全忽略在它里面的内容。很多时候我们也许不会注意到这个，特别是当父元素没有关联什么样式并且嵌套在它里面的子元素看起来是排列正确的时候。

当嵌套的元素没有整齐排列时，样式的错误就可能表现出来。看下面的例子.“.box-set”的div应该有一个淡灰色的背景，但因为嵌套在它里面的元素都是浮动的却导致看不到背景。查看“.box-set” div会发现它的高度是0.





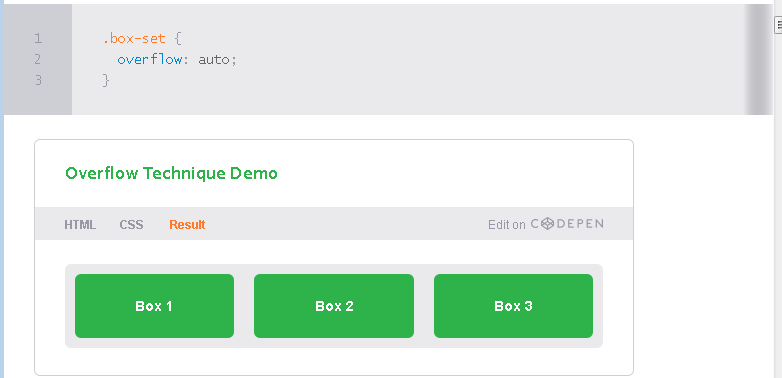
一个可以包含着这些浮动元素的方式是在父元素闭合标签前面放置一个空元素，但是需要是包含上样式描述clear:both.用这种方式可以清除浮动并且在大多数情况是有效的，但它没有确切的语义。当页面有很多浮动元素需要清除浮动时这种空元素的数量会开始迅速增加，并且这些元素对页面来说是没有任何语境意思的。

幸运的是我们有很多不同的方法用来包含这些浮动元素，最流行的方法包括Overflow技术和clearfix技术

### Overflow技术

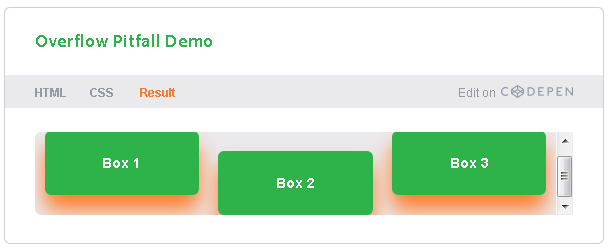
一项用父元素包含浮动元素的技术是使用overflow属性。把父元素的overflow属性值设置为auto就可以包含浮动，会给父元素一个实际的高度，在我们的例子里是包含一个灰色的背景。

要让这个在IE6里起作用需要为父元素设置高和宽。在苹果电脑的IE浏览器上使用overflow:auto 同样会给父元素加上滚动条，所以它更适合用overflow:hidden。



使用overflow技术也会带来一些缺点。举例来说，当添加样式或者移除父元素跨度外的嵌套元素时，比如试着添加边影和下拉菜单。下面的实例中，你可以看到父元素外围的边影是怎么被切除的。另外，第二个盒子在父元素的外边被裁减了。

不同的浏览器对overflow属性的处理是不一样的，并且在不同处理方式中也有滚动条的应用。看下面在不同浏览器中的同一个例子，观察在每个浏览器中这些列的不同展示。



### Clearfix技术

基于浮动元素的上下文语境一种更好包含浮动的技术也许是clearfix技术。Clearfix技术可能更复杂但确实比overflow技术有更好。

Clearfix 技术是基于在父元素上使用:before 和 :after伪元素。使用这些伪元素我们可以在包含浮动元素的父元素前面和后面创建隐藏的元素。:before 伪元素是用于防止子元素顶部的margin塌陷而使用样式display:table创建的一个表格单元元素。这些同样是可以在IE6和7上确保连贯的。:after 伪元素用于防止子元素底部margin的塌陷，同样清除嵌套的浮动。

特别在IE6和7中要给父元素添加\*zoom特性触发haslayout 机制，可以限定元素该怎么展示和限定出内容边界，也可以定义出有关联的元素是怎么交互的。

那上面提到的同一个例子来说你可以看到这些浮动元素是怎么被包含的并且这些元素可以在父元素外面存在。

.box-set:before,

.box-set:after {

content: "";

display: table;

}

.box-set:after {

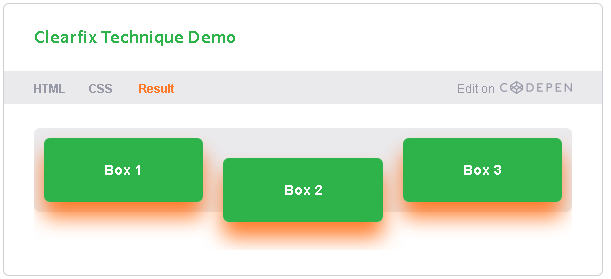
clear: both;

}

.box-set {

\*zoom: 1;

}



### 高效的包含浮动

使用哪种技术归根到底取决于当前的内容和你个人的喜好。有些人偏向于直接使用clearfix 技术因为它能总体上保持一致。而有些热觉得clearfix技术在一些实例中显得代码太多而倾向于基于内容的一种混合的技术。你要使用哪种取决于你，只要确保有完好的文档结构并且定义良好就可以了。

一个常用方法是给要包含浮动元素的父元素赋予一个class。用clearfix 作为例子，Dan Cederholm 帮助创建了类名为group。这个group类名就可以用于任何要包含浮动元素的父元素。

.group:before,

.group:after {

content: "";

display: table;

}

.group:after {

clear: both;

}

.group {

\*zoom: 1;

}

--单个伪元素—

目前单个的:before和:after伪元素是允许添加到前置元素上的，但这并没什么实际意义。当你尝试在用clearfix 技术时使用别的 :before和:after伪元素content可能得不到想要的结果。

在上面的例子里，clearfix的样式不能存在”box-set”类里面。另外，包含浮动元素的父元素里需要加上类”group”。

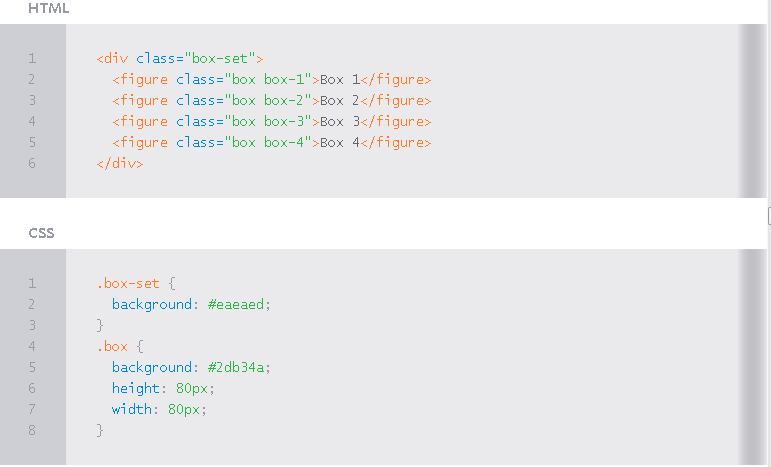
## --Position 属性--

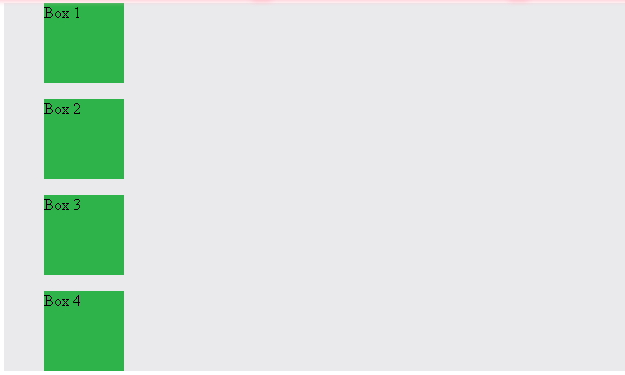
有时候你使用position属性控制元素时可以有比float更多操作性的。Position属性有五个不同的值，每一个值都提供了精确定位一个元素的不同方法。

### Position:Static

元素position的默认值是static,意味着它们没有或者接受任何特定的盒偏离属性(box offset)。此外，元素根据它们默认的状态会因目的不同而被放置(在不同位置)。

在下面的例子，所有的盒子都一个个从上到下堆砌着，因为它们都是块级别的元素并且没有float(浮动)到任何特定的方向。





### Position :relative

Position的relative属性和static属性很类似。最主要的区别是relative属性可以使用盒偏离值(box offset)的top、left、right和bottom。这些盒偏离属性可以帮助精确定位元素，并且让元素从默认位置偏离到任何方向上。

--盒偏离(box offset)是怎么工作的—

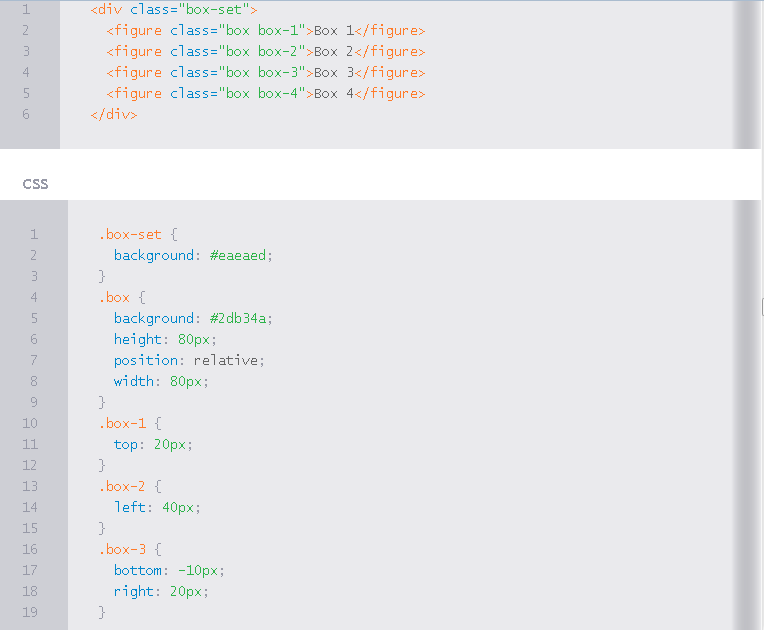
盒偏离的属性[top,left,right和left]，用于指定元素该怎么放置和往哪边方向放。这些偏离属性只适用于那些带有relative,absolute和fixed定位属性的元素。

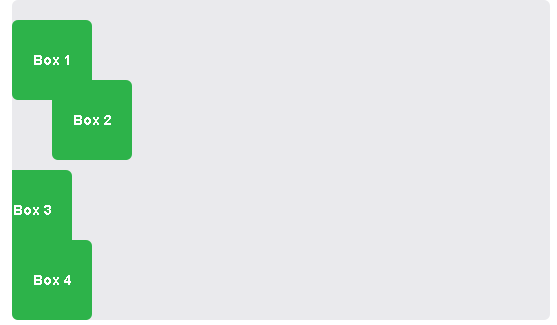
对于”relative”的元素，这些偏离属性指定元素相对它本来位置的偏离。举例来说，在一个“relative”元素上使用top:20px会把这个元素从它初始位置往下压20像素.如果把top 设为 -20px 则会把元素从初始位置往上推20个像素。

对于使用了“absolute”和“fixed”定位属性的元素这些偏离属性则指定的是元素和它父元素边界的距离。举例来说，在一个“absolute”元素上定义top:20px会把它从它带有“relative”定位的父元素顶部往下压20px.而把top 值改为 -20px 则会把元素从它相关联的父元素顶部被往上托出20像素。

带有relative定位属性的元素在盒偏离的时候依然是保持着正常的静态的页面流的。在这种情况下其他的元素不会影响到这个“relative”元素的初始位置。也正因为这样，这些带有”relative”的元素可能会因为其他的元素没有偏移原来的位置而出现覆盖或被遮挡。

在下面的例子中你将看到即便元素有通过偏离属性从它初始位置做出转移却仍然叠在其他元素的上面。这些偏离值使盒子一个覆盖另一个，而没有把每个放置到不同的的方向。当一个元素的position为relative时它周边的元素将仍旧处在它默认位置上。(参加box-3和box-4)





当在一个position为“relative”的元素上同时定义top和bottom两个偏离属性时，top属性将起主导作用。另外，如果在这”relative”定位元素上同时定义了left和right偏离属性，谁起主导作用要看这个页面是用什么语言编写的。例如，在英文编写的界面是left起主导作用，而用阿拉伯语编写的页面则是right起主导作用.(因为阿拉伯语是从右往左书写和阅读的)

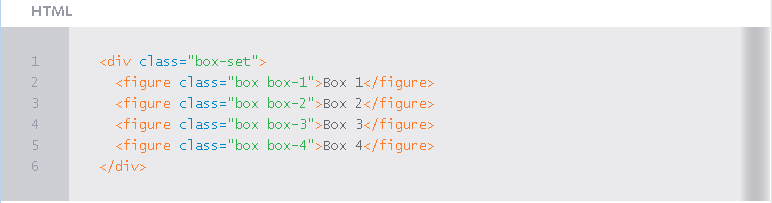
### Position: absolute

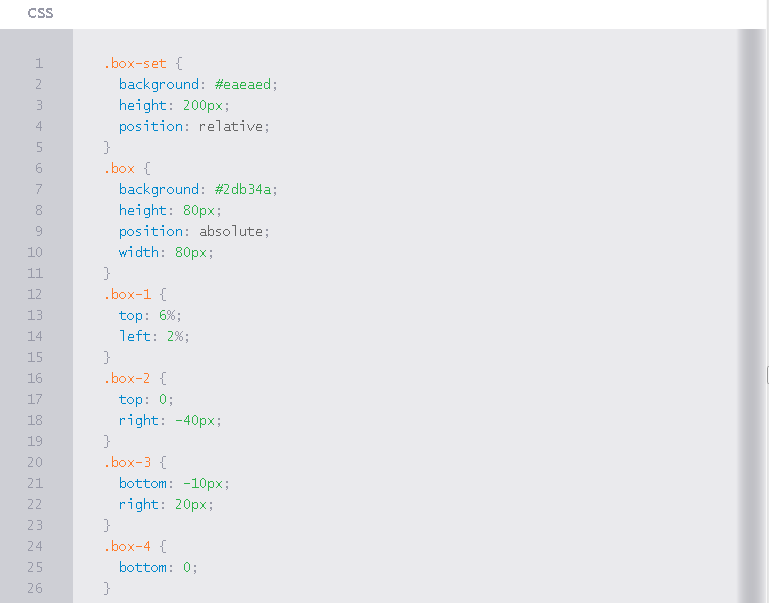
带有absolute定位的元素也接受盒偏离属性，但带有absolute的元素会脱离一般的文档流。由于与一般的文档流脱离，这类元素就直接根据包含它的带有relative或absolute属性的父元素来进行定位。如果这些元素没有这类的父元素，那么它们就会相对页面body来进行定位。

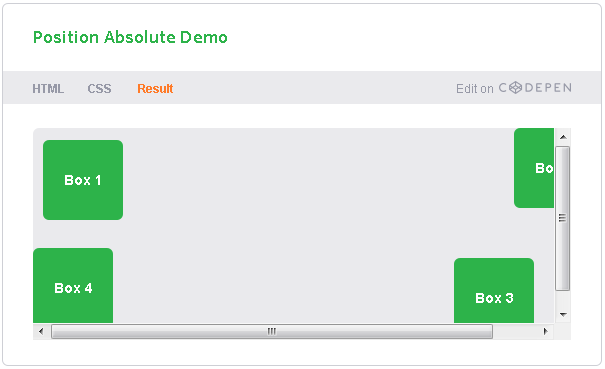
为使用了absolute定位的元素指定水平或垂直方向上的偏离值时会根据它父元素来进行移动。举例来说，一个带有top值为50px和right值为100px的元素会相对它的父元素向下偏离50像素同时离父元素右边界100像素。

还有，如果不为absolute的元素指定盒偏离值的话该元素会处在离它关联最密的元素的左上角。为元素设定一个偏离值，例如top，肯定会把元素在垂直方向上进行放置但在水平方向上仍是保持默认的左对齐。

在下面的例子你可以看到这些“box”元素是怎么通过absolute属性在与它相关联的relative父元素中定位的。每个单独的box通过一个正值被移动到特定的边界或者通过负值从指定的边界中被拖出来。







当一个元素是固定宽高并且是absolute定位的，同时对其使用top和bottom时top起主导作用。对于relative定位的元素，一个固定宽高的元素同时使用了left和right属性，谁起主导作用要看页面是用什么语言编写的。

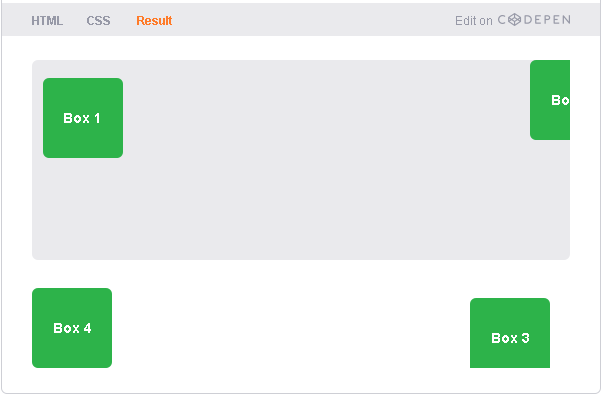
如果一个元素没有固定宽高并且是absolute定位的，对其同时使用top和bottom偏离属性会使该元素拉伸高度充满整个指定区域。如果是对其同时使用left和bottom属性会在水平方向拉伸宽度到整个区域。如果同时对其使用了四个属性则会让该元素占满整个区域。

### Position: Fixed

使用fixed进行定位所产生的作用和absolute很像，但fixed是相对浏览器窗口进行定位的，并且它不会随着页面的滚动而动。也就是说，无论用户滚动到页面哪个地方这类元素都一直在它原来的位置。 唯一要注意的是fixed属性不适用于IE6。如果想要在IE6中使用fixed定位需要使用适当的hack技术。

对fixed元素使用偏离值所产生的效果和absolute是一样的。

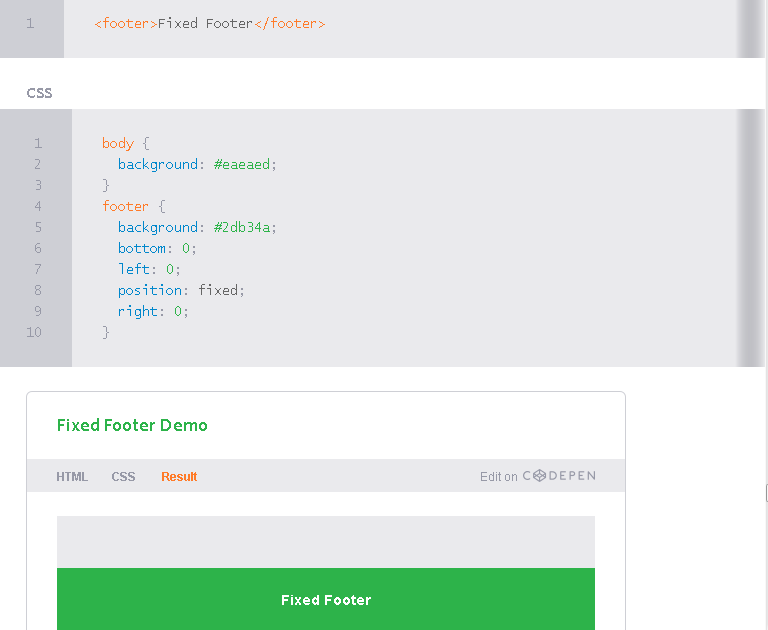
保持先前那个例子的偏离属性不变，观察它们相对浏览器窗口而不是它们父元素定位的效果。[把.box的position改为fixed]



### 固定头部和底部

一项关于fixed定位最常使用的操作是创建固定的头或尾然后固定在页面的一边。用以用户滚动(页面) 时的流动元素，并且用户通过视图窗口就可与之交互。

下面列出来的代码和例子展示了如果做到这样的。注意left和right偏离值是如何定义的。这个确保了footer能够占据页面整个底部宽度，并且它没有破坏盒模型同时允许自由使用margin，border和padding.



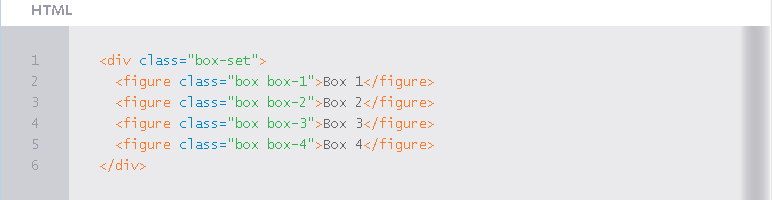
## --Z-Index 属性—

自然意义上的web页面通常被认为是有两个维度，即在x轴和y轴上显示元素。但当你对元素进行定位是他们有时出现一个元素在另一个的上方。要改变[这些元素是怎么堆叠](http://www.impressivewebs.com/a-detailed-look-at-the-z-index-css-property/)的顺序，也就是z轴，则要用到z-index属性。

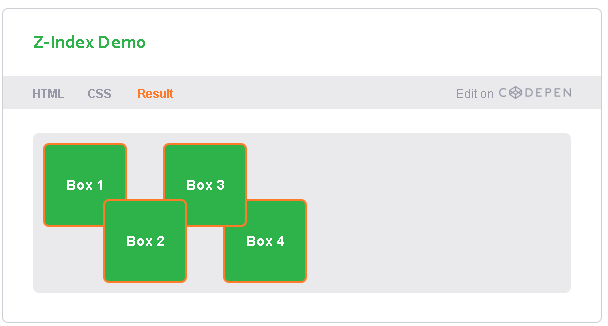
通常，这些出现在DOM里的元素放置在z轴上。在DOM意义上后来的元素是会出现在先前元素的上面的。使用z-index属性来改变这种堆叠顺序是很直接的。无论元素本身在DOM里是出于什么位置，若为其定义最高的z-index值就会出现在最上层。

要为一个元素应用z-index属性，必须先对其应用relative，absolute或者fixed。就像如果你要对元素应用偏离值一样。

在下面的例子中，如果不使用z-index属性那么每个box将准确的一个个放置着，即box2在box1上面，box3在box2上面而box4在box3上面。使用z-index 属性重新堆叠这些盒子让 box2 在所有box 上面，但是box3在 box2下面并且box4在box3下面。







# 3. Complex Selectors [复杂的选择器]

选择器是(或者不是)CSS里最重要的部分。它们形成级联并决定着怎样把样式应用到页面元素上。

到目前为止关于CSS方面的东西还没关注到选择器的。有时会有关于选择器规范的更新，但也从来没有非常重大的变更提升。幸运的是，后来也有更多关于选择器方面的，主要看看怎么选择不同的元素类型和元素在不同状态下的使用。

CSS3引进了新的选择符，为目前的操作展开了一个新的机会和提升的世界。在这里我们讨论选择器，不管新的旧的，并讲解怎么最好的使用它们。

## --常用的选择器—

在深入了解更多复杂的由css3带来的选择器之前，今天让我们先快速看看一些更常用的选择器。这些选择器包括类型 (type)，类(class)和ID选择器。

Type(标签)选择器是基于它的类型来定义一个元素，特别是这些元素是怎么通过HTML来定义的。Class(类)选择器是通过类的属性值来定义元素的，是一种在需要的时候把通用的样式重用到多种元素上的定义方法。最后呢，ID选择器则是基于ID值来定义元素的，是一种特殊的且只能在一个页面用一次的定义方法。

可参见以下示例:



#### Common Selectors Overview

| **Example** | **Classification** | **Explanation** |
| --- | --- | --- |
| h1 | Type Selector | Selects an element by it’s type |
| .tagline | Class Selector | Selects an element by the class attribute value, which may be reused multiple times per page |
| #intro | ID Selector | Selects an element by the ID attribute value, which is unique and to only be used once per page |

## --子选择器—

子选择器提供了选择在一个元素后面元素的方法，也就是父元素下的子元素。这些选择器可以用两种方法，使用后缀或直接选择子元素。

#### Child Selectors Overview

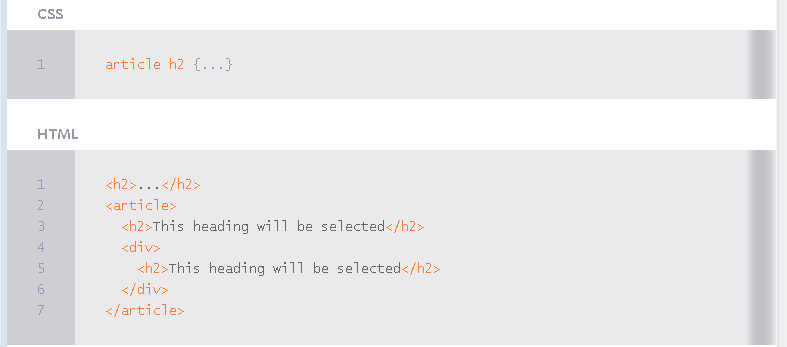
| **Example** | **Classification** | **Explanation** |
| --- | --- | --- |
| article h2 | Descendant Selector | Selects an element that resides anywhere within an identified ancestor element |
| article > p | Direct Child Selector | Selects an element that resides immediately inside an identified parent element |

### 后缀选择符

最常用的字选择器就是后缀选择器，用来匹配所有跟在祖先元素后的元素。后缀元素在文档树中不是像父子关系一样直接跟在祖先元素后面的，但也会在带有祖先元素的任何地方出现。选择器使用后缀选择符时是用空格来区分元素的，用来给元素序列建立一定的层次区分。

像article h2就是后缀选择器，只会选择在article元素里面的h2元素。但要注意，不管h2在article元素里的哪个位置，都会被选中。另外，在article元素外的h2元素就不会被选中了。

在下面例子中，article中的h2内容被选中。



### 直接子选择器

有时候后缀选择器会有一点过头了，会选择到超出预想的。当有时只要父元素下的直接子元素被选中时，而不是所有继承在祖先元素里的元素。这个时候就需要用一个更强大的符号>用在选择器的父元素和子元素中间。

举例来说，article>p就直指定义在article元素下的p元素。任何在article之外的P元素，或者嵌套在不是article元素里的别的元素，都不会被选上。

下面的例子中，你可以看到只有article里的P被选上，而Div里的不会被选中。



## --同级选择器—

知道怎么选择子元素是很有好处的，也是很常用的。但对于同级元素来说，它们在同一个父元素下，有时也可能被选中。同级选择器用于一般的兄弟节点和邻近的兄弟节点。

#### Sibling Selectors Overview

| **Example** | **Classification** | **Explanation** |
| --- | --- | --- |
| h2 ~ p | General Sibling Selector | Selects an element that follows anywhere after the prior element, in which both elements share the same parent |
| h2 + p | Adjacent Sibling Selector | Selects an element that follows directly after the prior element, in which both elements share the same parent |

### 一般的兄弟节点

一般的兄弟节点允许在他们兄弟元素基础上被选择，它们是在公共父元素下面。这类选择器是在两个元素之间使用波浪线~符号来实现的。第一个元素定义了与之兄弟关联的元素该是什么样的，并且它们必须拥有同一个父元素。

像h2~p选择器则是寻找和h2有同样父元素的且相关联的兄弟p元素。为了让p元素被选中，它必须跟在h2元素后面。

下例中的第5行和第9行会被选中因为在文档树里是跟在标题(h2元素)后面并且和它们兄弟关系的标题(h2元素)有同一个父元素(section).

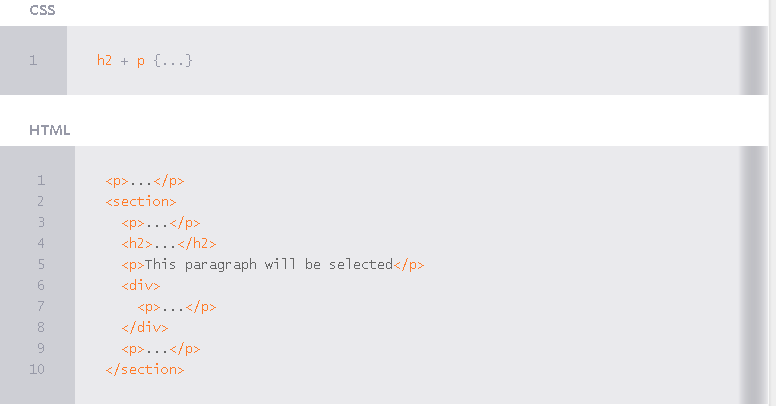


### 邻近的兄弟节点

有时候也许要更多的控制，就是选中直接在一个兄弟元素后的另一个兄弟元素，这就涉及到邻近节点元素的选择。邻近兄弟选择器会直接选择在另一个兄弟元素后面的元素。相比于一般兄弟选择器使用的波浪线，邻近兄弟选择器在两个元素之间使用+来分隔。同样，第一个元素决定了第二个元素是该怎么直接跟着和关联的，并且是拥有同一父元素的兄弟元素。

像h2+p选择器就只会选择直接跟在h2后面的p元素。并且它们是在同一个父元素下的。

在第5行的p元素会被选中因为它是直接和标题有兄弟关系并且在同一个父元素下的，所有被选中。



## --属性选择器—

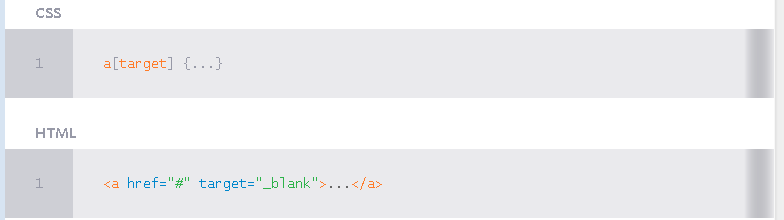
#### Attribute Selectors Overview

| **Example** | **Classification** | **Explanation** |
| --- | --- | --- |
| a[target] | Attribute Present Selector | Selects an element if the given attribute is present |
| a[href="http://google.com/"] | Attribute Equals Selector | Selects an element if the given attribute value exactly matches the value stated |
| a[href\*="login"] | Attribute Contains Selector | Selects an element if the given attribute value contains at least once instance of the value stated |
| a[href^="https://"] | Attribute Begins With Selector | Selects an element if the given attribute value begins with the value stated |
| a[href$=".pdf"] | Attribute Ends With Selector | Selects an element if the given attribute value ends with the value stated |
| a[rel~="tag"] | Attribute Spaced Selector | Selects an element if the given attribute value is whitespace-separated with one word being exactly as stated |
| a[lang|="en"] | Attribute Hyphenated Selector | Selects an element if the given attribute value is hyphen-separated and begins with the word stated |

一些常见的选择器在早期的时候也没看着是属性选择器，因为这些元素是基于它的类或者ID值的。这种用类和ID的属性选择器在最开始的时候有着很强大的作用。但另一些属性选择器这些年开始涌现，特别是随着CSS到了有了很大的飞跃。现在可以根据元素现有属性和可能包含的值来进行选择。

### 现有属性选择器

要介绍的第一个属性选择器定义了基于该元素是否包含一个属性，不管这个属性有没有实际的值去选择元素。要选择出一个有没有包含属性的元素仅仅在选择器上给需要找的属性加上中括号[] 就可以了。这个方括号跟不跟在一个标签或者类后面完全取决于你想要的多强的特异性等级。



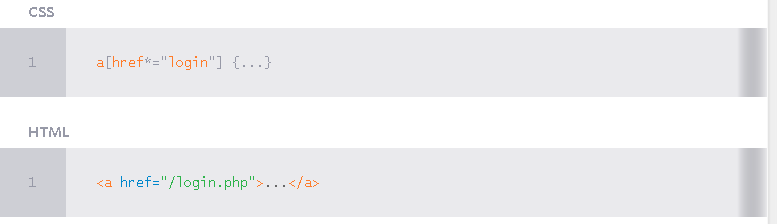
### 等值属性选择器

用特指，精确匹配，先前可能用到的同样选择器的值去定义元素，但这次在中括号里包含的是属性名和想要匹配的值。在中括号里属性名后面会跟上=和” ”并且在引号中是想要匹配的值。



### 包含属性选择器

当要根据属性值的部分内容来寻找元素，不是匹配所有内容时，这个星号\*也许可以在选择器的中括号中被用到。星号必须跟在属性名后面，在等号的前面。用来表示这个值只要出现了或者包含在属性值里面。



### 头部属性选择器

除了根据定义的值寻找是否包含在属性值里之外，根据属性值是以什么开头的也可以作为选择元素的条件。使用上尖号^把它放置在属性值和等号中间就可以匹配到以该值打头的属性值。

CSS:a[href^="https://"] {...} HTML:<a href="https://chase.com/">...</a>

### 结尾属性选择器

和头选择器相反，也有尾属性选择器。相比于使用上尖号，尾属性选择器使用美元符号$在选择器中括号中位于属性名和等号之间。使用美元符号表示要匹配的值是以该属性值为结尾的。

CSS:a[href$=".pdf"]{...}

HTML:

<a href="/docs/menu.pdf">...</a>

### 空格属性选择器

有时候属性可能是用空格分开的，使得只有一个符号需要在做选择器的时候被匹配到。这个时候使用波浪线~在选择器中括号的属性名和等号之间表示一个确切的用空白格分隔的属性值会被选到。

CSS：a[rel~="tag"] {...}

HTML:

<a href="#" rel="tagnofollow">...</a>

### 连字符属性选择器

当一个属性值使用连字符而不是空白符相连时，竖线|会被用在选择器中括号的属性名和等号中间。竖线表示属性值可能是连字符分隔的但是连字符的值必须是指定的值开头的。

CSS: a[lang|="en"] {...}

HTML: <a href="#" lang="en-US">...</a>

## --伪 类 –

在HTML中伪类和一般的类有点像但伪类不是通过标签直接指明，而是通过用户对文档的操作产生的一种动态的填充结果。你以前看到过最常用的一种伪类是 :hover。注意到它前面的冒号是所有伪类都会用到的。

### 链接型伪类

在链接里相对常用到的有两个基本伪类。:link 和:visited两个伪类用于链接是否被访问。:link 伪类用于没有被访问的锚点而 :visited伪类用于在浏览器历史记录里已经被访问过的链接的样式。

### 用户响应型伪类

根据用户的操作对同一个元素会应用上不同的伪类，包括 :hover, :active 和 :focus伪类。其中 :hover 伪类是在用户鼠标移动到元素上时被应用，大多数情况是用在锚点元素上的。而:active伪类则是用在用户要触发的元素上的，像是点击一个元素。最后，:focus伪类是用在用户在页面上要注意的点上的那个元素，通常是在使用键盘从一个元素切到另一个元素上时。

### 用户接口状态伪类

既然有关于链接方面的伪类同时也有关于用户元素接口状态的伪类，特别是针对表单元素。这些用户接口元素状态伪类包括 :enabled, :disabled, :checked 和 :indeterminate.

其中 :enabled伪类是用来选择在默认状态下被启用的input标签，而 :disabled伪类用来选择没有被启用的input标签。很多浏览器默认是灰显disabled的input标签用来告知用户这个input标签是不可用的，但如果用上:disabled伪类还是可以改变它们的样式状态的。

后两个用户接口元素状态伪类 :checked和 :indeterminate是用在多选框和单选按钮上的。 :checked 伪类会选择你希望或者已经选上的多选框项或者单选按钮。当复选框或者单选按钮都没有被选上或保持着未选中状态，使用 :indeterminate伪类就可以匹配到这些元素。

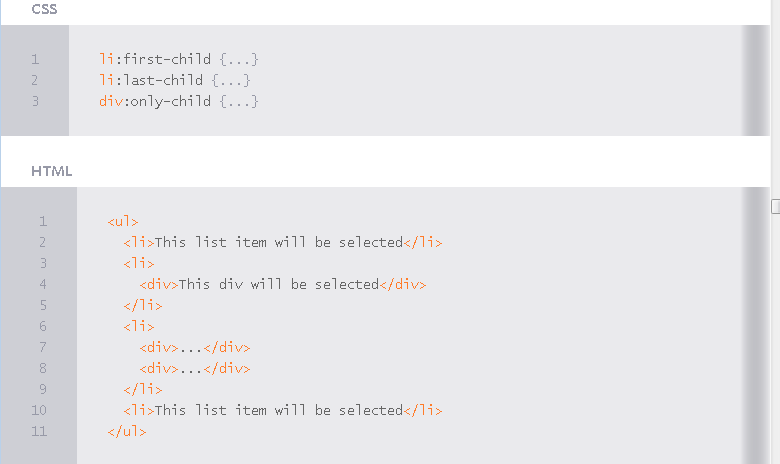
### 结构和位置型伪类

有一些伪类是基于结构和定位的，意思它们是根据元素在文档树的位置上决定样式的。这些基于结构和位置的伪类有着不同的样子和尺寸，每一个都有自己独特的功能。有些伪类比其他的要长些，而CSS3带来了全新的伪类集合用来补充到已经存在的伪类中。

#### :first-child, :last-child, & :only-child

第一个可能出现的基于结构和位置的伪类是 :first-child, :last-child和 :only-child. 其中 :first-child伪类用来确定一个元素是否是它父元素下的第一个子元素，而 :last-child伪类将会确定一个元素是否是父元素下最后一个子元素。这些伪类在列表中用来选择第一项或最后一项元素是最好用的。另外，:only-child伪类可以被写成 :first-child: last-child，但是 :only-child的差异性更小。

在这里li: first-child 定义了list下的第一项而 li: last-child 定义了list里的最后一项，所以第2行和第10行会被选中。选择器 div : only-child用来选择在父元素下只有一个子元素为div的元素，而没有其他兄弟节点的。这样的话第4行的元素会被选中因为它是指定列表项里唯一的div。



#### :first-of-type, :last-of-type, & :only-of-type

能够找到父元素下的第一，最后和唯一的元素确实挺有帮助，而且很多时候都用得到。但有时你只想找到指定类型的第一，最后或唯一的那个元素。譬如，你只想找到在article下面的第一或最后一个p元素，或者在article下面的唯一一个图片元素。幸运的是这正是 :first-of-type, :last-of-type和 :only-of-type可以用到的地方。

其中 :first-of-type伪类会选择父元素下该类型的第一个元素，而 :last-of-type伪类会选择父元素下该类型的最后一个元素。而 :only-of-type伪类会选择在父元素下该类型的唯一元素。

在下面的例子中p:fisrt-of-type和p:lats-of-type伪类会分别选择article中第一个和最后一个p元素，不管这两个p元素是不是article中的第一或最后一个子元素。根据这些选择器第三行和第6行会被选中。使用 img:only-of-type选择器可以选择第5行的图片元素因为它是article中出现的唯一图片元素，所以会被选中。



最后这里有一些使用数字或函数表达式来选择元素的基于结构和定位的伪类。这些伪类包括 :nth-child(n), :nth-last-child(n), :nth-of-type(n) 和 :nth-last-of-type(n).所有这些特殊的伪类都有 nth 作为前缀并且使用n这个参数可以在插入语中接受数字或表达式。

在插入语中的数字或表达式定义了选择哪个或哪几个元素会被选中。只使用数字会在文档树中从头或者从尾开始数单个的元素并选中一个，而使用表达式可以在文档树中从头部或者尾部开始数一定量的元素并且选中一组或多个元素。

#### 🡪在伪类中使用数字和表达式

像之前提到的，仅仅在伪类中使用数字会从文档树的起始或结束位置一个个数找到要选的元素。例如，使用 li:nth-child(4)选择器会选中在list下面的第四个元素。从第一个开始数而后每有一个list项就加一，直到最后定位并选中第四项。当仅仅使用数字时必须是正数。

伪类的表达式有这些形式：an,an+b,an-b,n+b,-n+b 和 –an+b。相同的表达式可以意为并读作(axn) ± b.其中a变量乘以倍数代表哪些元素会被选中而b变量代表从哪里开始数。

例如，li:nth-child(3n)选择器定义在list中三倍的list项。也就是说这个表达式等同于3x0,3x1,3x2然后一直下去。然后你会发现这个表达式可以选中像第三个，第六个和所有三倍的元素。

另外，关键字odd和even可能会被使用。如你所想，这些会根据要求选择奇数或偶数元素。所以有了关键值就不希望用2n+1表达式来选择奇数元素而2n来选择所有偶数元素。

选择器 li:nth-child(4n+7)会选择从第七项开始每过四个list的项。同样，使用这个表达式等同于 (4x0)+7,(4x1)+7,(4x2)+然后一直下去。这个表达式的结果就是像第7项，第11项，第15项并且之后所有每加四的元素都会被选中。

在变量n前面不使用数字的前缀会被看做是用了正数1.使用 li:nth-child(n+5)选择器会从第五个元素开始开始选择每个元素，前四个元素就不会被选了。这个表达式的结果就是(1x0)+5,(1x1)+,(1x2)+5及其之后的元素被选中。

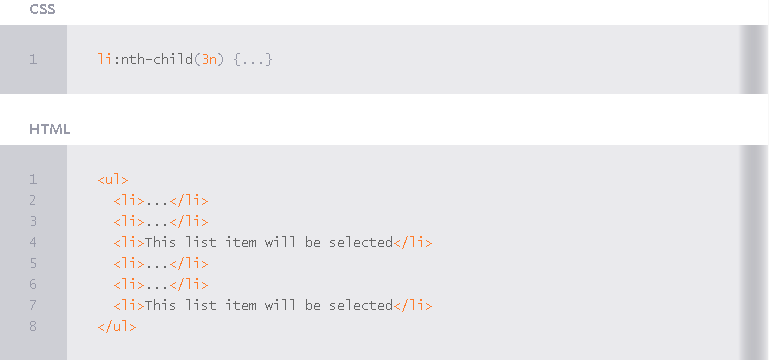
要让事情变得更复杂一点可能要用到负数。例如，选择器 li:nth-child(6n-4)会选择减去4之后的每六项元素，也就是会选中第2项，第8项和第14项并一直这样。和该选择器有相同效果的是li:nth-child(6n+2),就不用a-b这样的表达式了。

一个负的a变量或则a-n表达式，必须跟在a+b变量后面。当使用a-a变量或则 -n 参数做前导时b变量定义出会数到多高的值。例如，选择器li:nth-child(-3n+12)会选中前12项中的每三项元素。选择器li:nth-child(-n+9)则会因为n参数选择前9项元素,没有声明a变量，默认值就是-1.

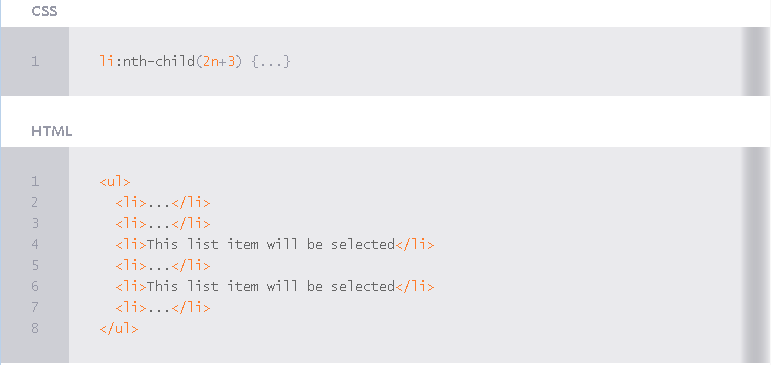
#### :nth-child(n) & :nth-last-child(n)

在大体了解了伪类中数字和表达式是怎么工作的之后让我们看看在实际的伪类中哪些数字和表达式会被用到，首先要提到的是 :nth-child(n)和 :nth-last-child(n)伪类。这个伪类在外观上同 :first-child和 :last-child伪类有点像，并且所有父元素下的元素和特别指定的元素会被选中。其中 :nth-child(n)会从文档树的头部开始寻找而 :nth-last-child(n)会从文档树的尾部开始工作。

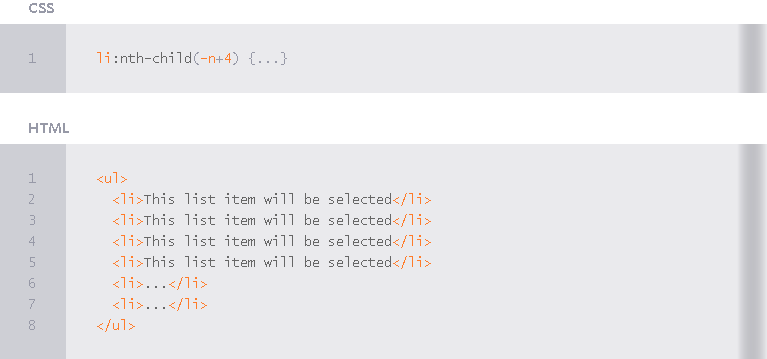
使用 :nth-child(n)伪类，观察li:nth-child(3n)选择器。这个定义的是每三个list项被选中,所以第4和第7行会被选中。



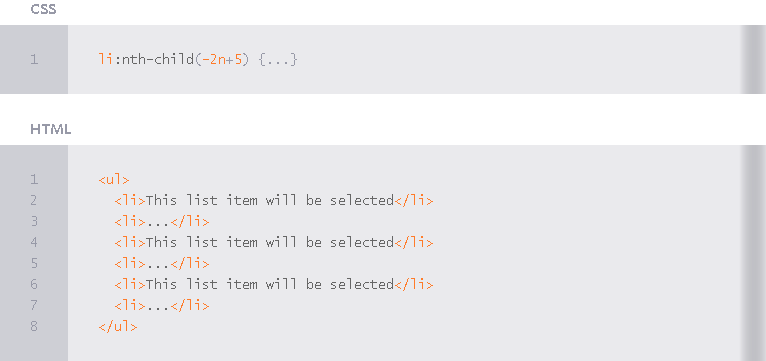
在 :nth-child(n)伪类中使用不同的表达式会产生不同的选择。举例来说，选择器li:nth-child(2n+3)会从第三个开始每两个list项被选中。结果，第4行和第6行会被选中。



再次改变表达式，这次使用负值，做出新的选择。选择器 li:nth-child(-n+4)定义了除前四项元素会被选中外其他的都不会被选中，所以下面第2到5行会被选中。



再在n参数前加一个负值。选择器li:nth-child(-2n+5)定义了在前五个项里每两个选择元素，就是第2,4和第6行会被选中。



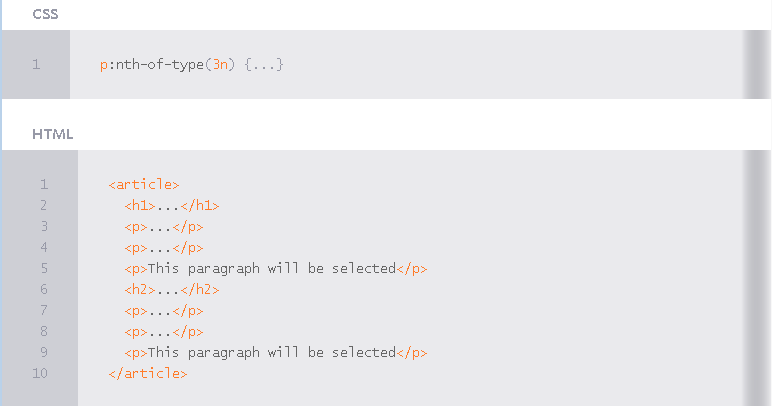
从 :nth-child(n)伪类转变到 :nth-last-child(n)伪类上会发生方向上的改变，使用:nth-last-child(n)伪类会从文档树的尾部开始计算元素。例如选择器 li:nth-last-child(3n+2)定义了从倒数第二个开始的每三个元素，一直到列表的开头。这里看到在第三和第六行的list项会被选中。



#### :nth-of-type(n) & :nth-last-of-type(n)

伪类 :nth-of-type(n), :nth-last-of-type(n)和伪类 :nth-child(n),:nth-last-child(n)很像，和伪类nth-child(n),:nth-last-child(n)根据父元素计算每个元素不同的是:nth-of-type(n), :nth-last-of-type(n)伪类只会根据他们自己的类型进行元素选择。例如，当要计算article下面的p元素时，伪类:nth-of-type(n), :nth-last-of-type(n)会忽略掉那些标题，div和其他一些不是p的元素，选出选择器指定要选出的元素，而nth-child(n),:nth-last-child(n)却会计算那些不是P的元素而不管它们的类型。

用到:nth-of-type(n)伪类的选择器 p:nth-of-type(3n)我们可以定义出父元素下的每三个p元素而不用去管父元素下的其他兄弟元素。在这里第5和第9行会被选中。



相比于伪类 :nth-childe(n)和 :nth-last-child(n)，在 :nth-of-type(n), :nth-last-of-type(n)伪类中最大的区别是 :nth-of-type(n)伪类是从文档树的起始部分寻找而 :nth-last-of-type(n)伪类是从文档树的底部开始查找。

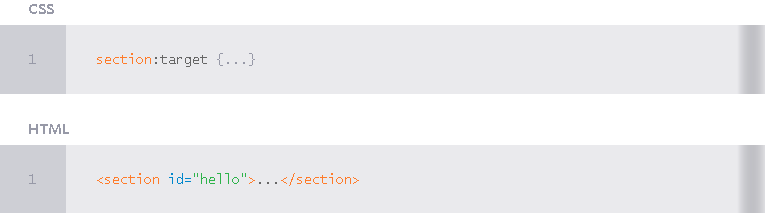
使用 :nth-last-of-type(n)伪类我们可以写出选择器 p:nth-last-of-type(2n+1)定义出从父元素中最后一个p元素开始往上寻找每两个p元素。在这里第4,7和9行的p元素被选中。



### target伪类

伪类 :target用于有ID属性值的URI片段标识的样式元素。使用URI的片段标识可以用哈希值，#直接跟在后面来做识别。URL <http://example.com/index.html#hello>中包含了”hello”这个片段。当在页面中标识符匹配到ID属性值，例如<section id=”hello”>，这个元素可以用 :target伪类进行定义和样式化。片段标识符在使用页面链接时更常用，或者在链接到页面中其他部分时。

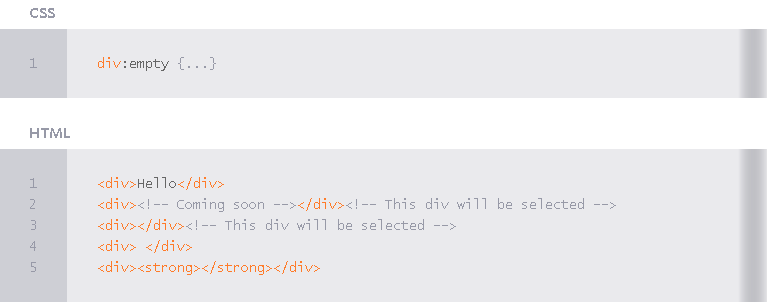
观察下面的代码，如果用户使用带有URI片段标识为#hello时通过使用:target伪类可以给带有相同id属性值的section进行样式化。如果uri的片段标识符改变了，并且匹配上另一个section的id属性值，这个新的section会通过使用同样的选择器和伪类进行样式化。



### empty 伪类

伪类 :empty允许选择不包含子节点或文本节点的元素。但是像comment，进程指令和空的文本节点是不被认作子元素所以不包含在内。

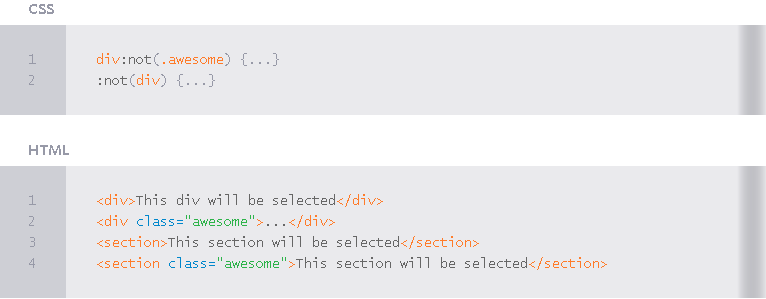
使用div:empty 伪类定义了不包含任何子元素或文本节点的div。在下面的第2,3行div会被选中因为它们安全是空的。即便第二个div包含了comment但是也不算做有子元素，所以该div还是空的。第一个div包含了文本，第三个div包含了一个空的文本空格，最后一个div包含了strong作为子元素，所以它们都不符合定义规则不会被选上。



### 否定型伪类

否定型伪类 :not(x)会调用一个参数用作选择器里的一个过滤器。选择器p:not(.intro)使用了否定型伪类会找出所有不包含类intro的p元素。这个选择器里p作为开头后面跟着 :not(x)伪类。在小括号里的是否定选择符，这里是用类.intro作为例子。

下面的选择器div:not(.awesome)和 :not(div)都使用了伪类 :not(x)。其中前者指向的是所有不包含类awesome的div而后者指向所有不是div的元素。结果就是第一行的div还有第3,4行的两个section被选中，它们都被标记上黑体。唯一没有被选中的是有类名为awesome的div，因为两个否定型伪类都没选中它。



## --伪元素—

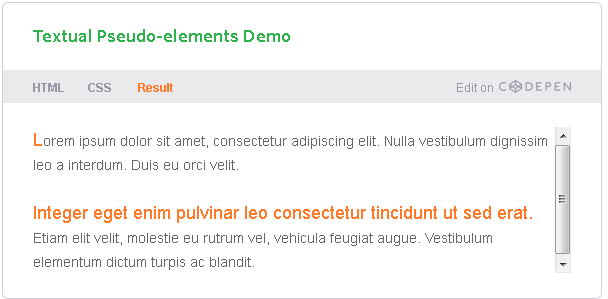
伪元素是不存在于文档树里的动态元素，而且当在选择器中使用这些伪元素时会允许在页面一些特别的地方附上样式。特别要注意的一点，一个选择器一次只能使用一个伪元素。

### 文本型伪元素

第一类要说的文本型伪元素是 :first-letter和 :first-line. 其中前者会指定元素中文本的第一个字母而后者会指定元素中文本的第一行内容。

在下面的例子中带有alpha类名的p元素的第一个文字和带有类名为bravo的p元素第一行文字会被赋上更大的字号并且设置成橘黄色(orange)。会出现这样的选择是因为它们分别使用了 文本型伪元素:first-letter和 :first-line.



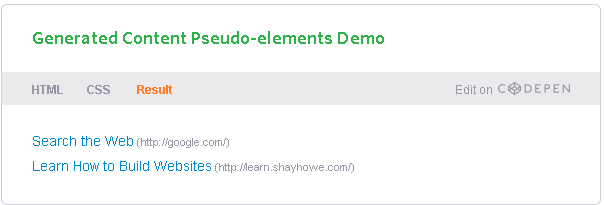


### 构建文本型伪元素

构建内容的伪元素 :before和 :after会在选中元素的内部新增一个内联级别的伪元素。这些伪元素更常用在关联content属性上用于在页面上添加不很重要的信息，但也不仅仅是用于这些地方。这些伪元素的其他用途是在页面上添加一些没有确切语义但不让页面混杂的用户接口组件元素。

其中 :before伪元素会在被选元素的前面新增一个伪元素而 :after会在被选元素的后面新增一个伪元素。这些伪元素嵌套在被选元素里面。下面的 :after伪元素使用圆括号在实际的链接后面显示出href属性值。这里这些信息是有用的，但是在一些不支持伪元素的浏览器中这些信息不是必要的。





### 片段型伪元素

片段型伪元素 ::selection定义为文档中的一部分(内容)通过用户的操作被选中或者高亮显示。选中的会被样式化，但只能使用color，background，background-color和text-shadow属性。其中background-image 属性会被忽略因为它在这没有意义。倒是background属性可以用来添加颜色，所有的图片都会被忽略掉。

#### 单冒号(:)对比双冒号(::)

片段型伪类是在CSS3中添加的并且为了和伪类做出区分所以给伪元素加了双冒号。幸好绝大多数浏览器都支持单冒号和双冒号，在伪元素里 ::selection就是以双冒号开头的。

在下面的例子中因为使用了 ::selection片段型伪类所有被选中的文本背景会显示为橙色并且文本阴影会被移除(一般的浏览器选中文字会是蓝底白字，现在变成橙底白字)。同样要注意，有Mozilla做前缀的片段型伪类 ::-moz-selection是为了确保得到所有浏览器的支持。



#### 浏览器对选择器的支持情况

使用CSS可以让选择器做出很多很棒的效果，但有时候它们得不到老浏览器的支持。在做出任何重要效果之前先检查你想用的选择器是否可以被大多数常用的浏览器访问，然后再去商讨这些工作是否必要。

CSS3.info提供了一个[CSS3 Selectors Test](http://tools.css3.info/selectors-test/test.html)工具用来你检查选择器是否可以被正在使用的浏览器支持。直接从供应商那里检查浏览器是否支持确实是不错的主意。

另外，[Selectivizr](http://selectivizr.com/)，一个JavaScript工具，为IE6-8使用这些选择器提供强大的支持。如果需要的话还可以从[jQuery selectors](http://api.jquery.com/category/selectors/) 获得更多支持。

#### 选择器的速度和性能

选择器的速度和性能是很值得关注的，如果使用了太多复杂的选择器会拖慢页面加载(渲染)。所以写的时候要注意并且当一个选择器看起来有点奇怪时要再想想看是否能找到更好的解决方法。