# CSS2.0教学讲义

# 什么是CSS

css是**c**ascading **s**tyle **s**heet 层叠式样式表的简写。用于修饰网页元素(标记)。

W3C规定尽量用CSS样式替代XHTML属性

现在的互联网前端三层：

HTML 超文本标记语言 从**语义**的角度描述页面**结构**。

CSS 层叠式样式表 从**审美**的角度负责页面**样式**。

JS JavaScript 从**交互**的角度描述页面**行为**。

|  |
| --- |
| 1. <style type="text/css"> 2. p{ 3. color:red; 4. font-size: 30px; 5. text-decoration: underline; 6. font-weight: bold; 7. text-align: center; 8. font-style: italic; 9. } 10. h1{ 11. color:blue; 12. font-size: 50px; 13. font-weight: bold; 14. background-color: pink; 15. } 16. </style> |



我们写css的地方是style标签，就是“样式”的意思，写在head里面。

后面的课程你将知道，css也可以写在单独的文件里面，现在我们先写在style标签里面

|  |
| --- |
| 1. <style type=”text/css”> 2. </style> |

type表示“类型” ， text就是“纯文本”。css也是纯文本的。

**css对换行不敏感，对空格也不敏感。但是一定要有标准的语法。冒号，分号都不能省略。**

语法：

|  |
| --- |
| 1. 选择器{ 2. k:v; 3. k:v; 4. k:v; 5. k:v; 6. } 7. 选择器{ 8. k:v; 9. k:v; 10. k:v; 11. k:v; 12. } |

# 一些常见的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 赋值 | 介绍 |
| color | color: red; | 前景色（文字颜色） |
| font-size | font-size: 20px; | 设置文字大小 |
| width | width: 30px; | 设置宽度 |
| height | height:30px; | 设置高度 |
| background-color | background-color:red; | 设置背景颜色 |
| text-align | text-align:center; left|right; | 文字居中 |
| text-indent | text-indent: 2em； | 首行缩进 |
|  |  |  |
|  |  |  |

# 选择器的分类：

☞基础选择器

◆标签选择器

◆类选择器 （☆☆☆☆☆）

◆id选择器

◆通配符选择器

☞复合选择器

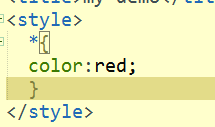
◆标签指定式选择器 （☆☆☆☆）

◆后代选择器 （☆☆☆☆☆）

◆并集选择器

## 通配符选择器

\*{...}(自动应用于所有标记中)



## 标签选择器

标签：就是html中的标签。

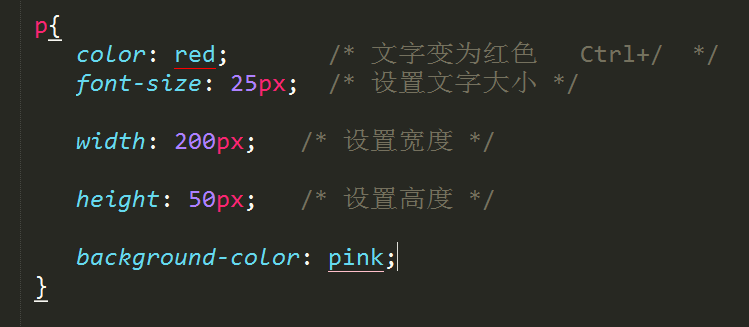
1） 所有标签都能够当做选择器，比如body、h1、dl、ul、span等等

2） 不管这个标签藏的多深，都能够被选择上。

3） 选择的是所有的，而不是某一个。所以是共性，而不是特性。

用法：

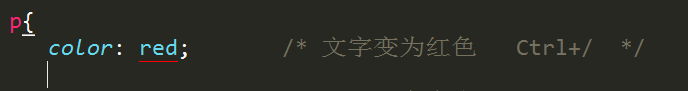
标签名{属性:值; 属性:值;}



补充：

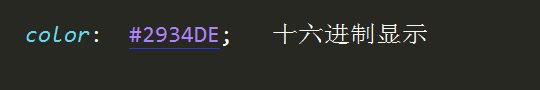
颜色的显示方式：

☞ 直接设置对应颜色的名称



☞ 通过十六进制显示

以#开头 0-ff



☞通过rgb方式显示

R: 代表的是红色（red） 0-255

G: 代表的是绿色（green） 0-255

B: 代表的是蓝色（blue） 0-255

☞rgba

a: 设置透明度 alpha 0-1

常见的颜色：

红色 #f00 rgb(255,0,0)

绿色 #0f0 rgb(0,255,0)

蓝色 #00f rgb(0,0,255)

白色 #fff rgb(255,255,255)

黑色 #000 rgb(0,0,0)

黄色 #ff0 rgb(255,255,0)

浅灰色 #eee rgb(200,200,200)

深灰色 #333 rgb(3,3,3)

## id选择器

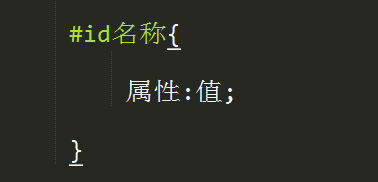
#表示选择id

|  |
| --- |
| 1. #lj1{ 2. font-size: 60px; 3. font-weight: bold; 4. color:black; 5. } |

1）任何的标签都可以有id，id的命名要以字母开头，可以有数字、下划线。大小写严格区别，也就是说mm和MM是两个不同的id。

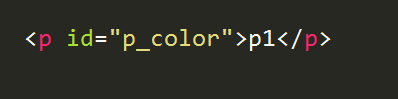
2）同一个页面内id不能重复，即使不一样的标签，也不能是相同的id。

也就是说，如果有一个p的id叫做haha，这个页面内，其他所有的元素的id都不能叫做haha。



调用：

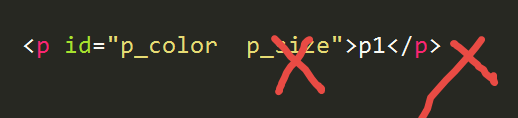
通过id属性调用样式



特点：

☞不建议多个标签使用同一id选择器的样式（id和js配合使用）

☞一个标签只能调用一个id样式



## 类选择器

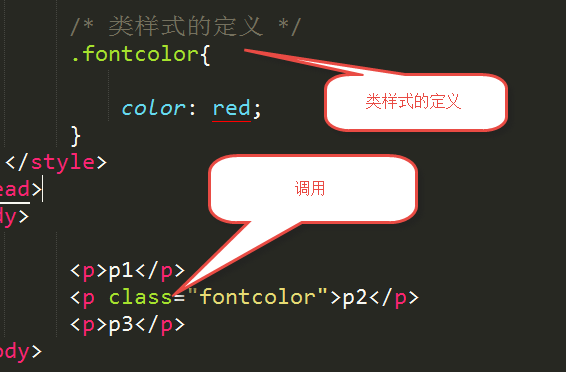
**.就是类的符号**。类的英语叫做class。

所谓的类，就是class属性，class属性和id非常相似，**任何的标签都可以携带class属性**。

.自定名称{属性:值;}

调用：

标签通过class属性调用类样式



特点：

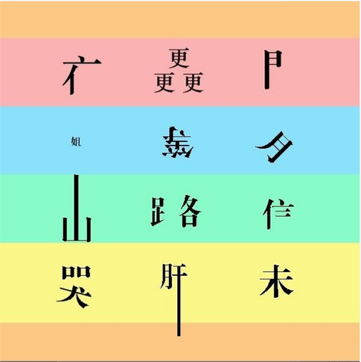
☞ 谁调用谁改变，多个标签可以同时调用一个类样式

☞ 一个标签可以调用多个类样式（相当于人的名字）



### 类的命名规范





夜半三更门半开；

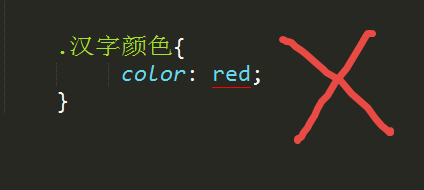
小姐等到月儿弯；

山高路断无口信；

哭断肝肠人未来。

☞不能以纯数字或者以数字开头定义类名

☞不推荐使用汉字定义类名



☞不能以特殊符号或者以特殊符号开头（“\_”除外）定义类名

☞不建议使用标签名或者属性名定义类名

**class属性可以重复，比如，页面上可能有很多标签都有teshu这个类**：

|  |
| --- |
| 1. <h3>我是一个h3啊</h3> 2. <h3 **class="teshu"**>我是一个h3啊</h3> 3. <h3>我是一个h3啊</h3> 4. <p>我是一个段落啊</p> 5. <p **class="teshu"**>我是一个段落啊</p> 6. <p **class="teshu"**>我是一个段落啊</p> |

css里面用.来表示类：

|  |
| --- |
| 1. **.**teshu{ 2. color: red; 3. } |

**同一个标签，可能同时属于多个类，用空格隔开**：

|  |
| --- |
| 1. <h3 class="**teshu zhongyao**">我是一个h3啊</h3> |

这样，这个h3**就同时属于teshu类，也同时属于zhongyao类**。

初学者常见的错误，就是写成了两个class：

|  |
| --- |
| 1. <h3 **class="teshu" class="zhongyao"**>我是一个h3啊</h3> |

所以要总结两条：

1） class可以重复，也就是说，同一个页面上可能有多个标签同时属于某一个类；

2） 同一个标签可以同时携带多个类。

类的使用，能够决定一个人的css水平。

比如，我们现在要做一个页面：

段落1

段落2

段落3

正确的思路，就是用所谓“公共类”的思路，就是我们类就是提供“公共服务”，比如有绿、大、线，一旦携带这个类名，就有相应的样式变化：

|  |
| --- |
| 1. <style type="text/css"> 2. .lv{ 3. color:green; 4. } 5. .da{ 6. font-size: 60px; 7. } 8. .xian{ 9. text-decoration: underline; 10. } 11. </style> |

每个标签，就去选取自己想要的类：

|  |
| --- |
| 1. <p **class="lv da"**>段落1</p> 2. <p **class="lv xian"**>段落2</p> 3. <p **class="da xian"**>段落3</p> |

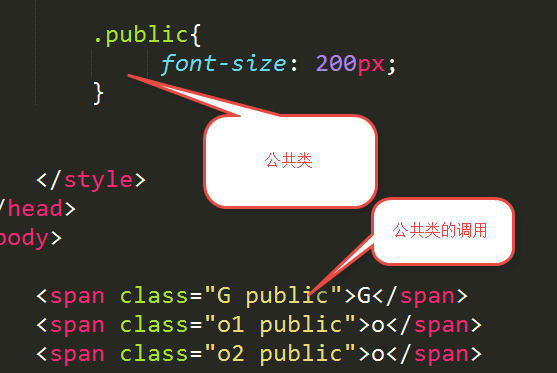
也就是说：

1） 不要去试图用一个类名，把某个标签的所有样式写完。这个标签要多携带几个类，共同造成这个标签的样式。

2） 每一个类要尽可能小，有“公共”的概念，能够让更多的标签使用。

补充：

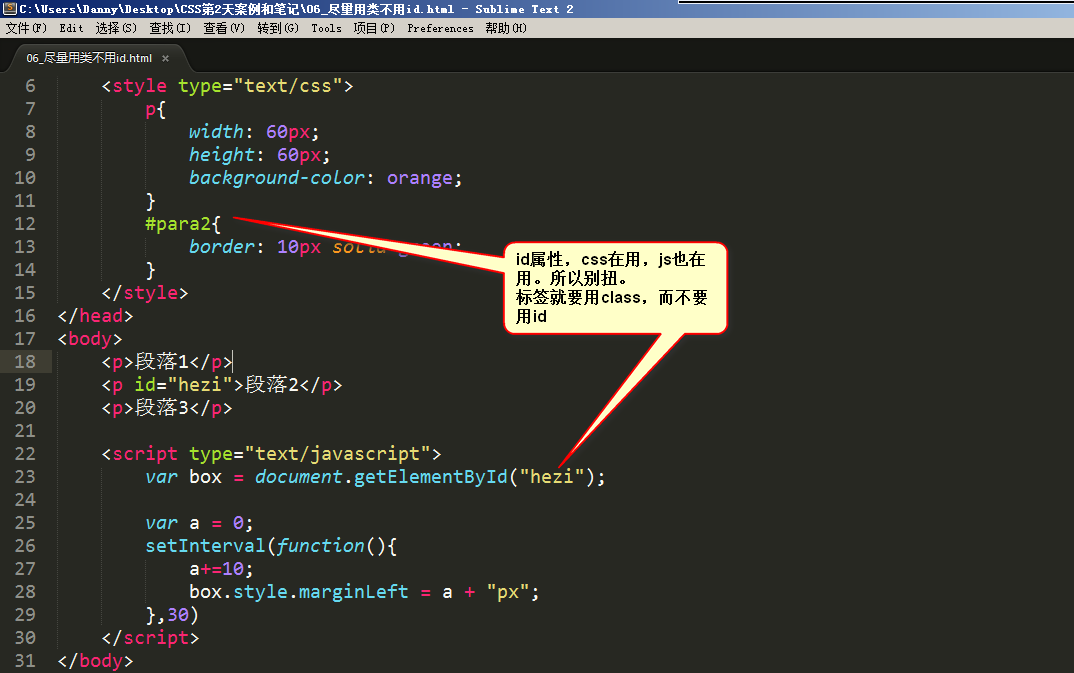
在写样式的时候（类选择器），如果有样式代码重复使用的时候，我们要提取公共类。



到底用id还是用class？

答案：尽可能的用class，除非极特殊的情况可以用id。

原因：id是js用的。也就是说，js要通过id属性得到标签，所以我们css层面尽量不用id，要不然js就很别扭。另一层面，我们会认为一个有id的元素，有动态效果。

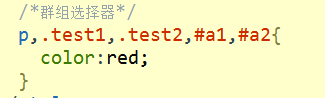


类上样式，id上行为

就是一个标签，可以同时被多种选择器选择，标签选择器、id选择器、类选择器。这些选择器都可以选择上同一个标签，从而影响样式，这就是css的cascading“层叠式”的第一层含义。

## 群组选择器

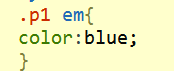
选择器名称,选择器名称...{...}(不同选择器但属性相同的缩写)



## 后代选择器

选择器名称 选择器名称{...}

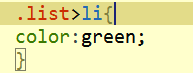
(两个选择器至少是父子关系)



## 子代选择器

选择器名称>选择器名称{...}

(两个选择器必须是父子关系)



## 伪类选择器

(1)行为伪类

a标签有4种伪类，要求背诵：

|  |
| --- |
| 1. a**:link**{ 2. color:red; 3. } 4. a**:visited**{ 5. color:orange; 6. } 7. a**:hover**{ 8. color:green; 9. } 10. a**:active**{ 11. color:black; 12. } |

a{}与a:link{}实现的效果是一样的

a:link 表示， 用户没有点击过这个链接的样式。 是英语“链接”的意思。

a:visited 表示， 用户访问过了这个链接的样式。 是英语“访问过的”的意思。

a:hover 表示， 用户鼠标悬停的时候链接的样式。 是英语“悬停”的意思。

a:active 表示， 用户用鼠标点击这个链接，但是不松手，此刻的样式。 是英语“激活”的意思。

a:focus 表示， 获取焦点（光标）的时候的样式

|  |  |  |  |
| --- | --- | --- | --- |
| a:link | a:visited | a:hover | a:active |

记住，这四种状态，在css中，必须按照固定的顺序写：

a:**l**ink 、a:**v**isited 、a:**h**over 、a:**a**ctive

如果不按照顺序，那么将失效。“爱恨准则”**l**o**v**e **ha**te。必须先爱，后恨。

:focus： 设置鼠标获得焦点状态

### 超级链接的美化

a标签在使用的时候，非常的难。因为不仅仅要控制a这个盒子，也要控制它的伪类。

我们一定要将a标签写在前面，:link、:visited、:hover、:active这些伪类写在后面。

**a标签中，描述盒子； 伪类中描述文字的样式、背景。**

|  |
| --- |
| 1. .nav ul li **a**{ 2. display: block; 3. width: 120px; 4. height: 40px; 5. } 6. .nav ul li **a:link** ,.nav ul li a**:visited**{ 7. text-decoration: none; 8. background-color: yellowgreen; 9. color:white; 10. } 11. .nav ul li **a:hover**{ 12. background-color: purple; 13. font-weight: bold; 14. color:yellow; 15. } |

text-decoration: None | underline | line-through

记住，所有的a不继承text、font这些东西。因为a自己有一个伪类的权重。

最标准的，就是把link、visited、hover都要写。但是前端开发工程师在大量的实践中，发现不写link、visited浏览器也挺兼容。所以这些“老油条”们，就把a标签简化了：

a:link、a:visited都是可以省略的，简写在a标签里面。也就是说，a标签涵盖了link、visited的状态。

|  |
| --- |
| 1. .nav ul li a{ 2. display: block; 3. width: 120px; 4. height: 50px; 5. text-decoration: none; 6. background-color: purple; 7. color:white; 8. } 9. .nav ul li a:hover{ 10. background-color: orange; 11. } |

(2)UI元素伪类

:checked 设置元素被选中状态(css3.0)

:disabled 设置元素禁止状态

:enabled 设置元素启用状态

## **伪元素选择器**

:before :设置元素之前的内容或图像

:after ： 设置元素之后的内容或图像

例如:

:before{

content:"内容"|url(图像)

}

:after{

content:"内容"|url(图像)

}

# Css书写位置介绍

## 内嵌式写法

<style type="text/css">

……

</style>

## 外联式写法

<link rel=”stylesheet” href=””>

☞新建一个文件，后缀名以.css命名（css文件）

☞在html页面中通过

<link rel=”stylesheet” href=””> 标签将css文件引入

<link type="text/css" rel="stylesheet" href="CSS文档URL" media="all"/>

说明:

type:目标文档类型

rel :当前文档和目标文档的关系

href :CSS文档URL

media:根据不同的媒介(设备)，

使用不同CSS样式

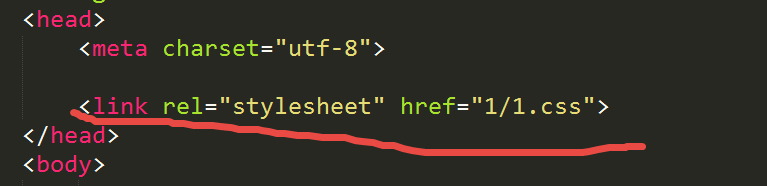
media:

all 所有

print 打印机

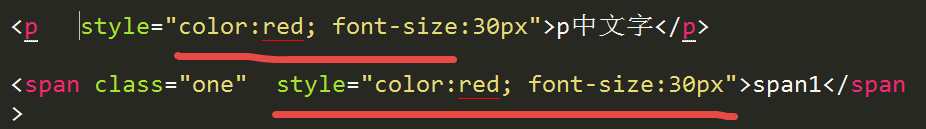
screen 屏幕

TV 电视



## 行内式写法

☞通过给标签设置style属性来设置样式。



优先级：

内联样式最高，内部和外部优先级同级后面的覆盖前面的样式。就近原则。

## Css书写方式之间的区别

☞内嵌式写法：

◆代码可维护性较差，没有实现css代码与html结构的完全分离

◆影响的范围只有当前页面

☞行内是写法：

◆代码可维护性极差，css代码和html结构没有实现分离

◆影响的范围只有当前标签

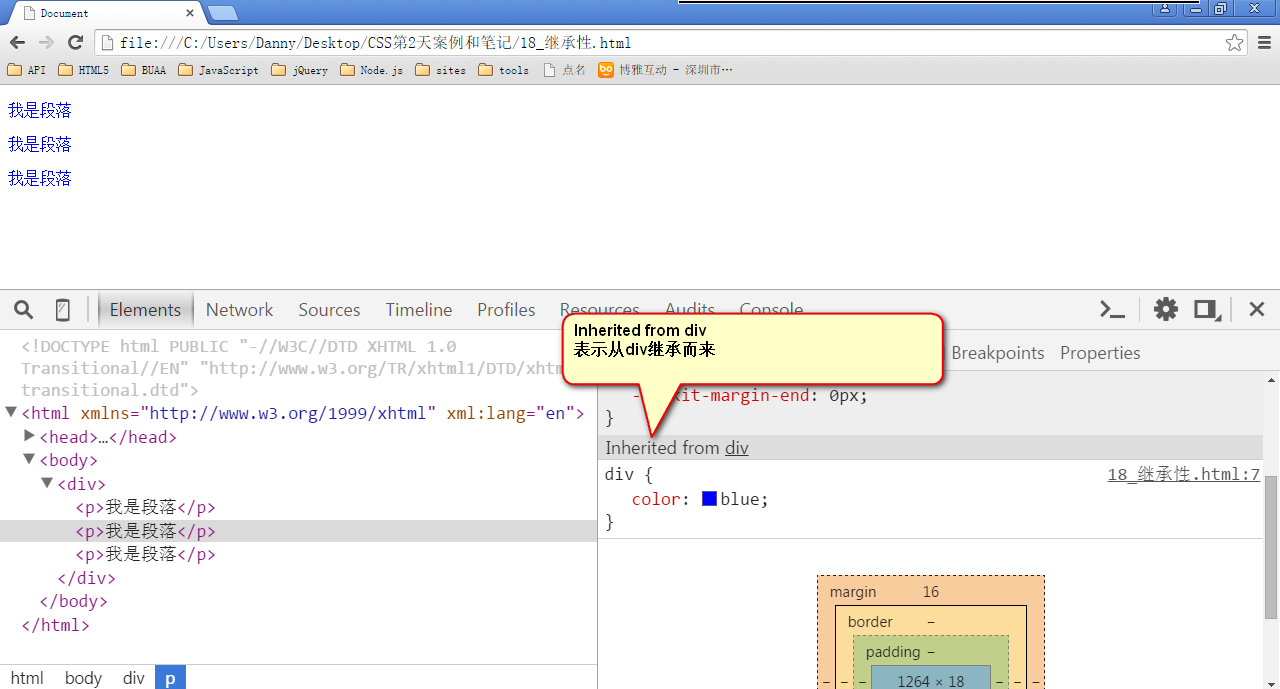
☞外联式写法：

◆代码可维护性高，css与html结构完全分离

◆影响范围广，当前整个网页站点。

# CSS的继承性和层叠性

## 继承性



有一些属性，当给自己设置的时候，自己的后代都继承上了，这个就是继承性。

继承性发生的前提是标签之间属于一种嵌套关系

☞文字颜色可以实现继承

☞文字大小可以实现继承

☞字体可以实现继承

☞行高可以实现继承

与文字有关的属性都可以 实现继承

哪些属性能继承？

color、 text-开头的、line-开头的、font-开头的。

**这些关于文字样式的，都能够继承； 所有关于盒子的、定位的、布局的属性都不能继承。**

特殊性：

<a href=”#”></a> 不能继承父元素中的文字颜色（层叠掉了）

<h1></h1> 标题标签不能继承父元素中的文字大小

所以，如果我们的页面的文字，都是灰色，都是14px。那么就可以利用继承性：

|  |
| --- |
| 1. body{ 2. color:gray; 3. font-size:14px; 4. } |

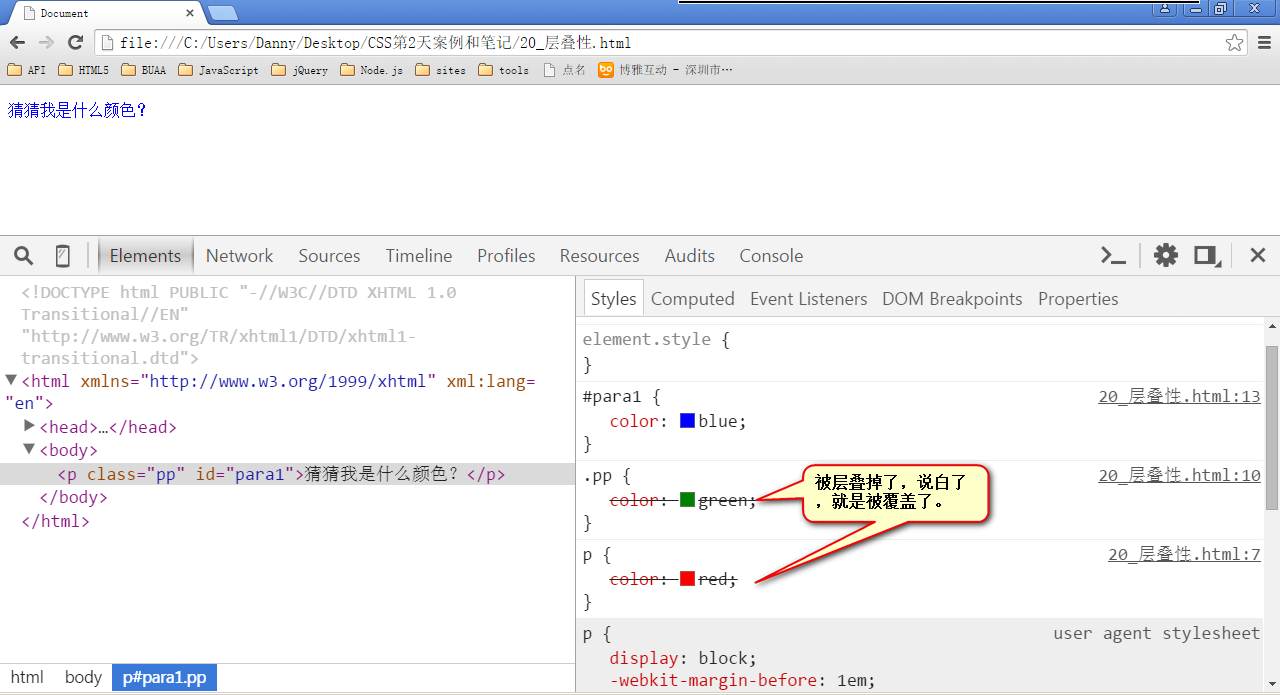
继承性是从自己开始，直到最小的元素。

## 层叠性

很多公司如果要笔试，那么一定会考层叠性。

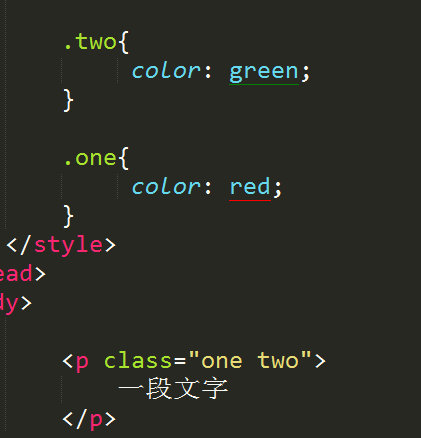
**层叠性：就是css处理冲突的能力。 所有的权重计算，没有任何兼容问题！**

**CSS像艺术家一样优雅，像工程师一样严谨。**



样式的覆盖。 样式的层叠性与样式的调用顺序没有关系，与样式的定义顺序有关。

层叠性发生的前提： 样式冲突



## 优先级

默认样式<标签选择器<类选择器<id选择器<行内样式<!important

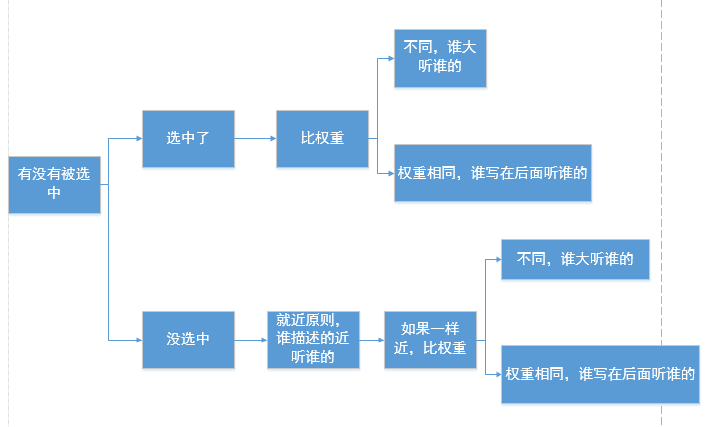
0 1 10 100 1000 1000以上

#### 优先级的特点

继承的权重为0

#### 权重会叠加

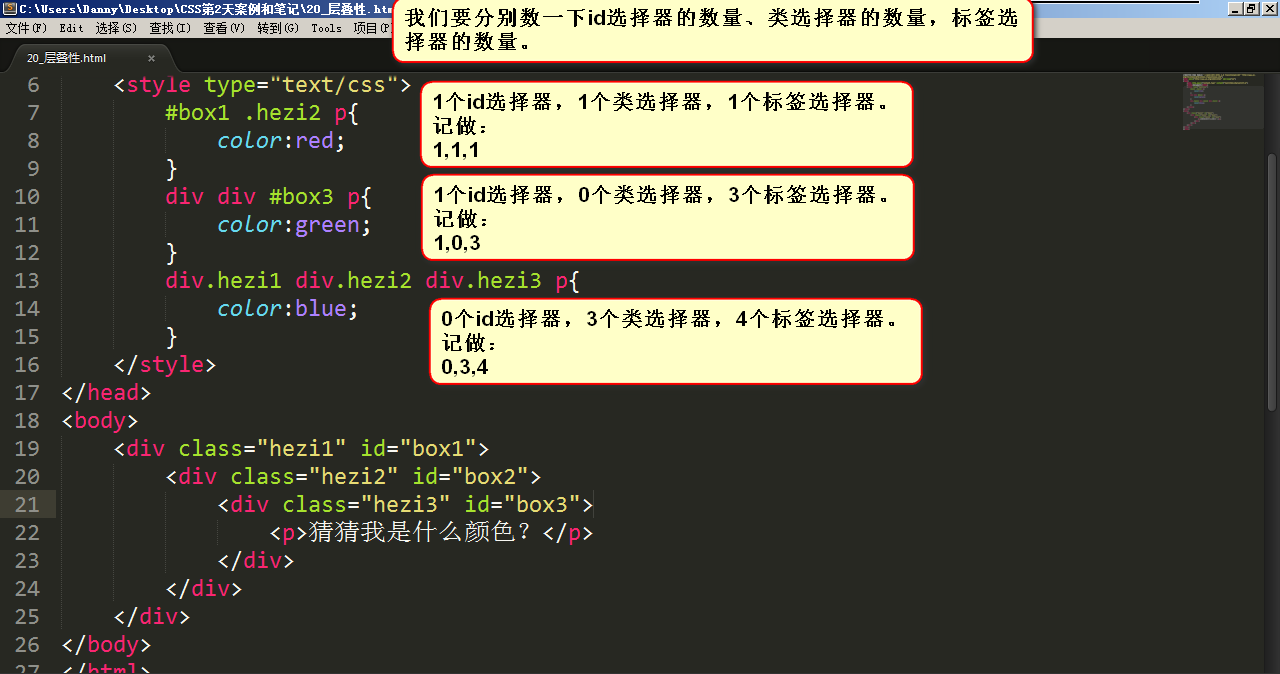
## 权重计算的总结



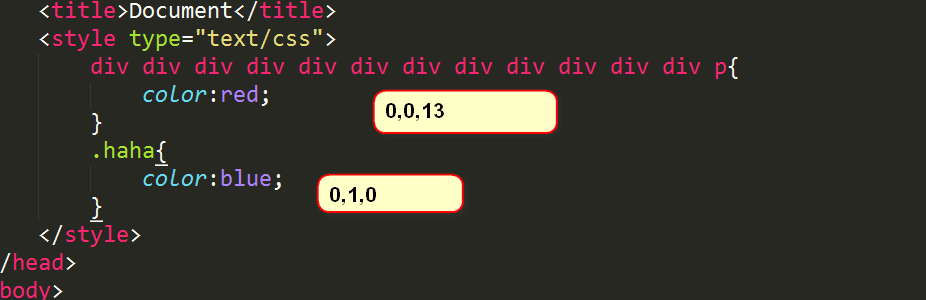
还要知道 ! important 的性质。

当选择器，选择上了某个元素的时候，那么要这么统计权重：

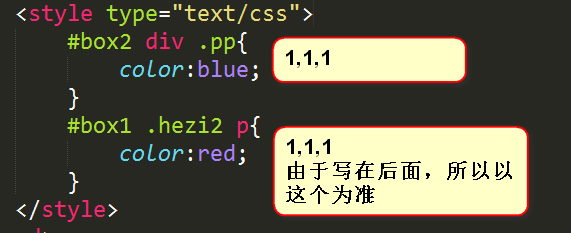
id的数量，类的数量，标签的数量



不进位，实际上能进位（奇淫知识点：255个标签，等于1个类名）但是没有实战意义！



如果权重一样，那么以后出现的为准：



================================我是华丽丽的分界线=================================

如果不能直接选中某个元素，通过继承性影响的话，那么权重是0。



**如果大家都是0，那么有一个就近原则：谁描述的近，听谁的。**



权重问题大总结：

1） 先看有没有选中，如果选中了，那么以（id数，类数，标签数）来计权重。谁大听谁的。如果都一样，听后写的为准。

2） 如果没有选中，那么权重是0。如果大家都是0，就近原则。

题目演示：

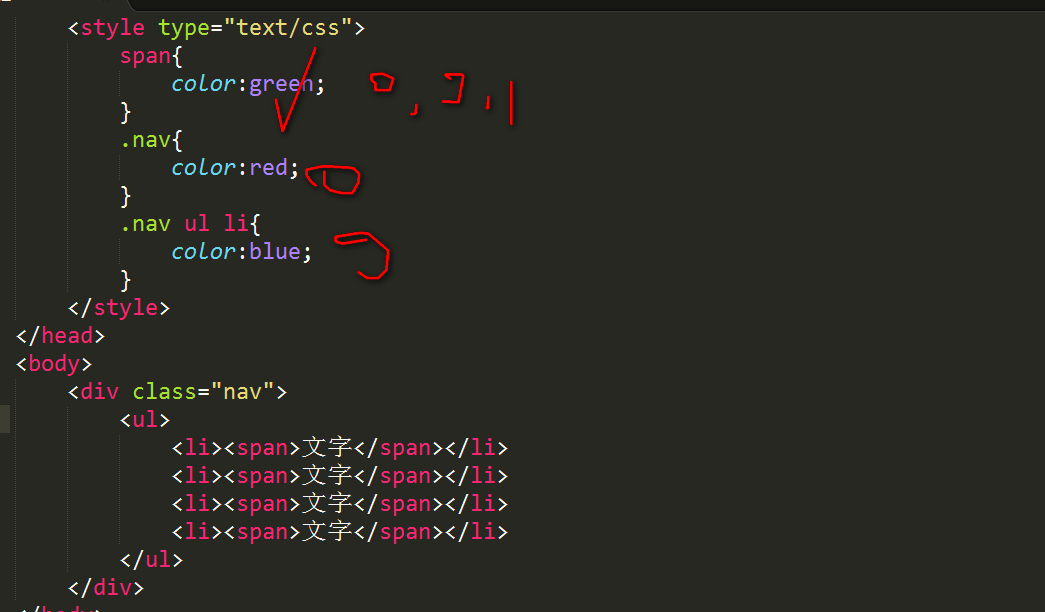
第1题：



第2题：



第3题：



第4题：



## c同一个标签，携带了多个类名，有冲突：

|  |
| --- |
| 1. <p class="**spec1 spec2**">我是什么颜色？</p> 2. <p class="**spec2 spec1**">我是什么颜色？</p> |

和在标签中的挂类名的书序无关，只和css的顺序有关：

|  |
| --- |
| 1. .spec2{ 2. color:blue; 3. } 4. .spec1{ 5. color:red; 6. } 7. </style> |

红色的。因为css中red写在后面。



## !important标记

|  |
| --- |
| 1. <style type="text/css"> 2. p{ 3. color:red **!important**; 4. } 5. #para1{ 6. color:blue; 7. } 8. .spec{ 9. color:green; 10. } 11. </style> |

important是英语里面的“重要的”的意思。我们可以通过语法：

|  |
| --- |
| 1. k:v !important; |

来给一个属性提高权重。这个属性的权重就是无穷大。

一定要注意语法：

正确的：

|  |
| --- |
| 1. font-size:60px **!important**; |

错误的：

|  |
| --- |
| 1. font-size:60px; !important; → 不能把!important写在外面 2. font-size:60px important; → 不能忘记感叹号 |

!important需要强调3点：

1） !important提升的是一个属性，而不是一个选择器

|  |
| --- |
| 1. p{ 2. color:red **!important**; → 只写了这一个!important，所以就字体颜色属性提升权重 3. font-size: 100px ; → 这条属性没有写!important，所以没有提升权重 4. } 5. #para1{ 6. color:blue; 7. font-size: 50px; 8. } 9. .spec{ 10. color:green; 11. font-size: 20px; 12. } |

所以，综合来看，字体颜色是red（听important的）；字号是50px（听id的）;

2） !important无法提升继承的权重，该是0还是0

比如HTML结构：

|  |
| --- |
| 1. <div> 2. <p>哈哈哈哈哈哈哈哈</p> 3. </div> |

有CSS样式：

|  |
| --- |
| 1. div{ 2. color:red **!important**; 3. } 4. p{ 5. color:blue; 6. } |

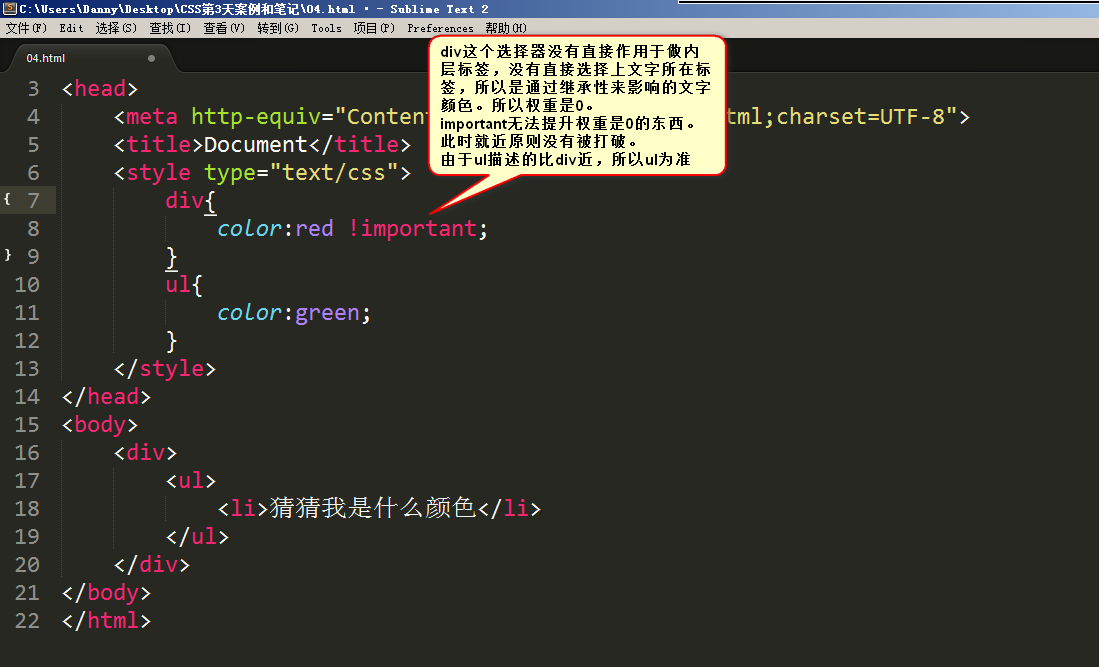
由于div是通过继承性来影响文字颜色的，所以!important无法提升它的权重，权重依然是0。

干不过p标签，因为p标签是实实在在选中了，所以字是蓝色的（以p为准）。

3）!important不影响就近原则

如果大家都是继承来的，按理说应该按照“就近原则”，那么important能否影响就近原则呢？

答案是：不影响。远的，永远是远的。不能给远的写一个important，干掉近的。



**! important做站的时候，不允许使用。因为会让css写的很乱。**

现在，我们知道层叠性能比较很多东西：

选择器的写法权重，谁离的近，谁写在下面。

# 盒模型

## 盒子中的区域

一个盒子中主要的属性就5个：width、height、padding、border、margin。

width是“宽度”的意思，CSS中width指的是内容的宽度，而不是盒子的宽度。

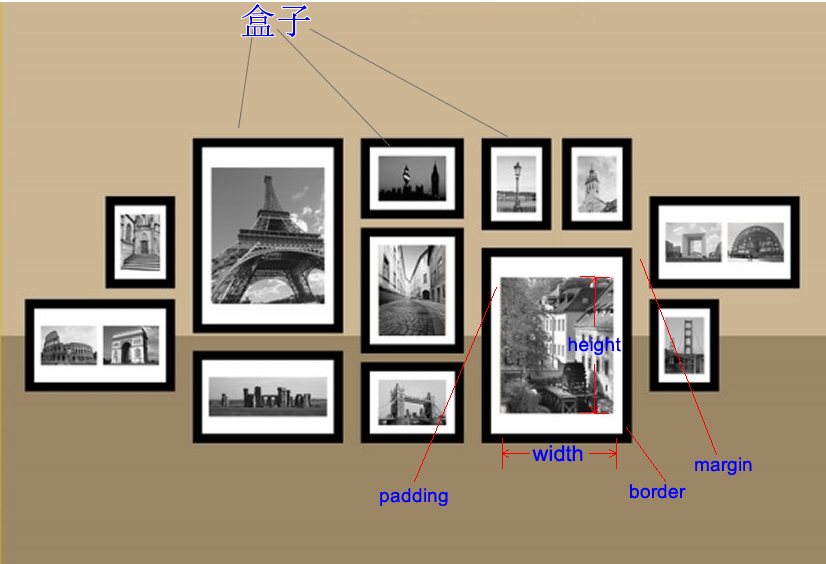
height是“高度”的意思，CSS中height指的是内容的高度，而不是盒子的高度

padding是“内边距”的意思

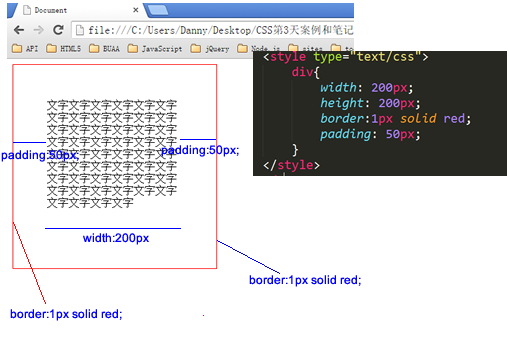
border是“边框”

margin是“外边距”

盒模型的示意图：



代码演示：



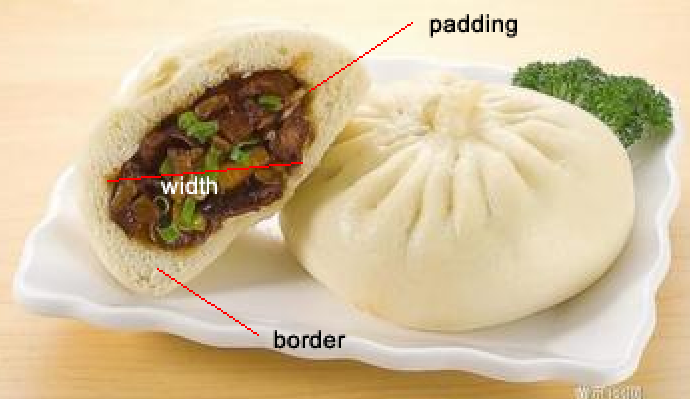
**这个盒子width:200px; height:200px; 但是真实占有的宽高是302\*302。 这是因为还要加上padding、border。**

**宽度和真实占有宽度，不是一个概念！！**

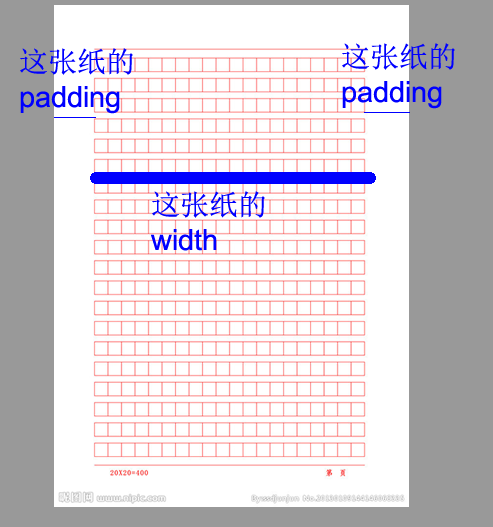
## 认识width、height

一定要知道，在前端开发工程师眼中，世界中的一切都是不同的：

比如丈量一个包子多宽？前端开发工程师，只会丈量包子馅：



丈量稿纸，前端开发工程师只会丈量内容宽度：



丈量人脸，只会丈量五官：



下面这两个盒子，真实占有宽高，完全相同，都是302\*302：

|  |
| --- |
| 1. .box1{ 2. width: 100px; 3. height: 100px; 4. padding: 100px; 5. border: 1px solid red; 6. } 7. .box2{ 8. width: 250px; 9. height: 250px; 10. padding: 25px; 11. border:1px solid red; 12. } |

**真实占有宽度= 左border + 左padding + width + 右padding + 右border**

这两个盒子的盒模型图，见下表：

|  |  |
| --- | --- |
|  |  |

小练习，大家自己写三个402\*402的盒子

**答案（答案有无穷多种，我们只写其中三种）：**

|  |
| --- |
| 1. .box1{ 2. width: 400px; 3. height: 400px; 4. border: 1px solid red; 5. } 6. .box2{ 7. width: 200px; 8. height: 200px; 9. border: 6px solid red; 10. padding: 95px; 11. } 12. .box3{ 13. width: 0px; 14. height: 0px; 15. padding: 200px; 16. border: 1px solid red; 17. } |

这三个盒子的盒模型图：

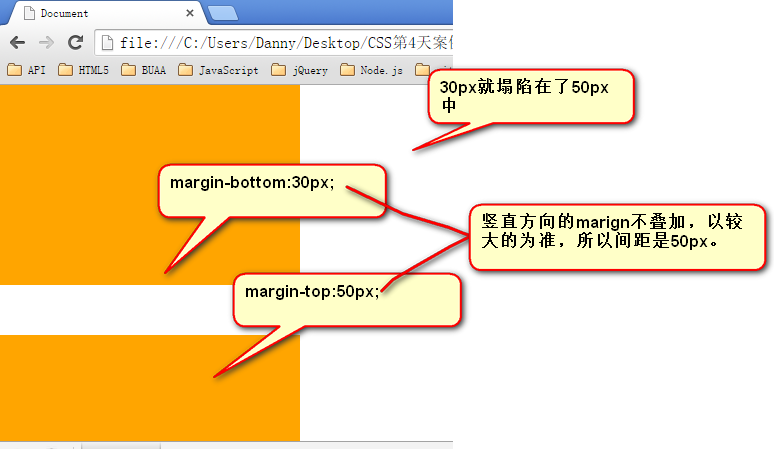
|  |  |  |
| --- | --- | --- |
|  |  |  |

如果想保持一个盒子的真是占有宽度不变，那么加width就要减padding。加padding就要减width。

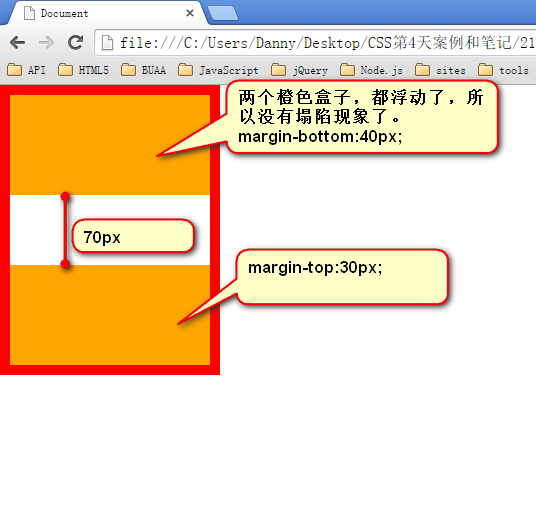
## margin

### margin的塌陷现象

**标准文档流中，竖直方向的margin不叠加，以较大的为准。**



如果不在标准流，比如盒子都浮动了，那么两个盒子之间是没有塌陷现象的：



### 盒子居中margin:0 auto;

margin的值可以为auto，表示自动。当left、right两个方向，都是auto的时候，盒子居中了：

|  |
| --- |
| 1. margin-left: auto; 2. margin-right: auto; |

简写为

|  |
| --- |
| 1. margin:0 auto; |

注意：

1） 使用margin:0 auto; 的盒子，必须有width，有明确的width

2） 只有标准流的盒子，才能使用margin:0 auto; 居中。

也就是说，当一个盒子浮动了、绝对定位了、固定定位了，都不能使用margin:0 auto;

3） margin:0 auto;是在居中盒子，不是居中文本。

文本的居中，要使用

|  |
| --- |
| 1. text-align:center; |

|  |
| --- |
| 1. margin:0 auto; → 让这个div自己在大容器中居中。 2. text-align: center; → 让这个div内部的文本居中。 |

普及一下知识，text-align还有

|  |
| --- |
| 1. text-align:left; 没啥用，因为默认居左 2. text-align:right; 文本居右 |

### 善于使用父亲的padding，而不是儿子的margin

如果父亲没有border，那么儿子的margin实际上踹的是“流”，踹的是这“行”。所以，父亲整体也掉下来了

这个p有一个margin-top踹父亲，试图将自己下移

|  |
| --- |
| 1. <div> 2. <p></p> 3. </div> |

结果：

|  |  |
| --- | --- |
|  |  |

**margin这个属性，本质上描述的是兄弟和兄弟之间的距离； 最好不要用这个marign表达父子之间的距离。**

所以，我们一定要善于使用父亲的padding，而不是儿子的margin。

### 关于margin的IE6兼容问题

IE6双倍margin bug

**当出现连续浮动的元素，携带和浮动方向相同的margin时，队首的元素，会双倍marign。**

|  |
| --- |
| 1. <ul> 2. <li></li> 3. <li></li> 4. <li></li> 5. </ul> |



解决方案：

1）使浮动的方向和margin的方向，相反。

所以，你就会发现，我们特别喜欢，浮动的方向和margin的方向相反。并且，前端开发工程师，把这个当做习惯了。

|  |
| --- |
| 1. float: left; 2. margin-right: 40px; |

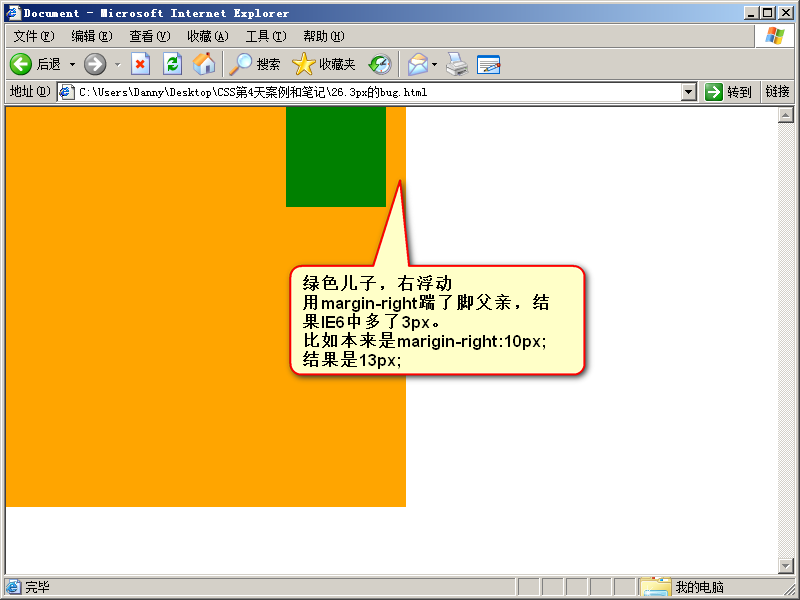
2）使用hack（没必要，别惯着这个IE6）

单独给队首的元素，写一个一半的margin

|  |
| --- |
| 1. <li class="no1"></li> |

|  |
| --- |
| 1. ul li.no1{ 2. \_margin-left:20px; 3. } |

IE6的3px bug



解决办法：

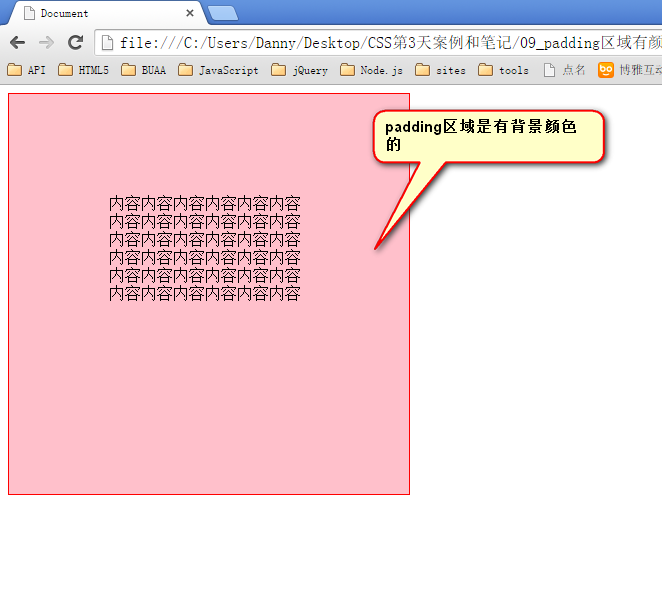
不用管，因为根本就不允许用儿子踹父亲。所以，如果你出现了3px bug，说明你的代码不标准。

**IE6，千万不要跟他死坑、较劲，它不配。 格调要高，老师给你讲，就是为了增加面试的成功率。不是为了让你成为IE6的专家。**

## padding

padding就是内边距。padding的区域有背景颜色，css2.1前提下，并且背景颜色一定和内容区域的相同。

也就是说，background-color将填充所有boder以内的区域。



padding是4个方向的，所以我们能够分别描述4个方向的padding。

方法有两种，第一种写小属性；第二种写综合属性，用空格隔开。

小属性：

|  |
| --- |
| 1. padding-top: 30px; 2. padding-right: 20px; 3. padding-bottom: 40px; 4. padding-left: 100px; |

top上、right右、bottom下、left左。

这种属性，就是复合属性。比如不写padding-left那么就是没有左内边距。

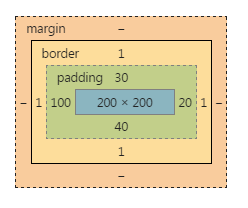
快捷键就是pdt、pdr、pdb、pdl 然后按tab。

综合属性：

如果写了4个值：

|  |
| --- |
| 1. padding:30px 20px 40px 100px; |

上、右、下、左



空格隔开的，四个数字就是上、右、下、左。

也就是说，前端开发工程师眼中的顺序不一样。

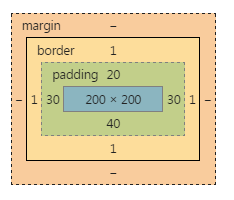
老百姓：上下左右

强调开发工程师：上、右、下、左

如果只写3个值：

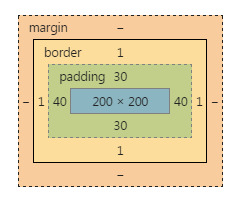
|  |
| --- |
| 1. padding: 20px 30px 40px; |

上、右、下、??和右一样



如果只写2个值：

|  |
| --- |
| 1. padding: 30px 40px; |



也就是说，

|  |
| --- |
| 1. padding: 30px 40px; |

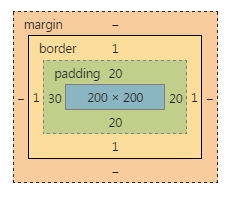
等价于：

|  |
| --- |
| 1. padding-top: 30px; 2. padding-bottom: 30px; 3. padding-left: 40px; 4. padding-right: 40px; |

要懂得，用小属性层叠大属性：

|  |
| --- |
| 1. padding: 20px; 2. padding-left: 30px; |

对应的盒模型图：



下面的写法错误：

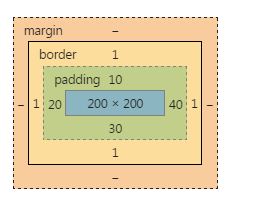
|  |
| --- |
| 1. padding-left: 30px; 2. padding: 20px; |

不能把小属性，写在大属性前面。

下面的题，会做了，说明你听懂了：

题目1，说出下面盒子真实占有宽高，并画出盒模型图：

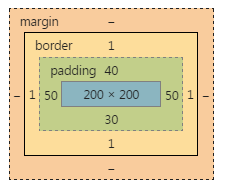
|  |
| --- |
| 1. div{ 2. width: 200px; 3. height: 200px; 4. padding: 10px 20px 30px; 5. padding-right: 40px; 6. border: 1px solid #000; 7. } |

 真实占有宽度 = 200 + 20 + 40 + 1 + 1 = 262px

题目2，说出下面盒子真实占有宽高，并画出盒模型图：

|  |
| --- |
| 1. div{ 2. width: 200px; 3. height: 200px; 4. padding-left: 10px; 5. padding-right: 20px; 6. padding:40px 50px 60px; 7. padding-bottom: 30px; 8. border: 1px solid #000; 9. } |

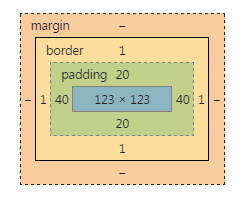
padding-left:10px； 和padding-right:20px; 没用，因为后面的padding大属性，层叠掉了他们。



强调一点，padding-left 不是padding-left-width

|  |
| --- |
| 1. padding-left:10px; √ 2. **padding-left-width:30px; ×** |

第3题，我现在给你盒模型图，请写出代码，试着用最最简单的方法写



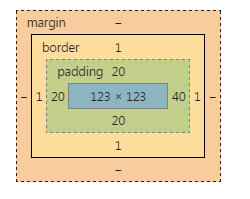
width:123px;

height:123px;

padding:20px 40px;

border:1px solid red;

第4题：



width:123px;

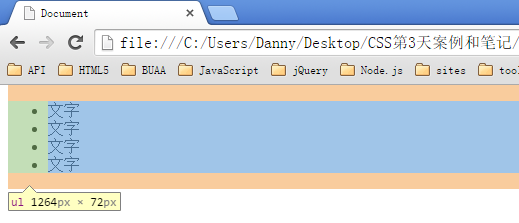
height:123px;

padding:20px;

padding-right:40px;

border:1px solid red;

一些元素，默认带有padding，比如ul标签。



所以，我们为了做站的时候，便于控制，总是喜欢清除这个默认的padding：

|  |
| --- |
| 1. \*{ 2. margin: 0; 3. padding: 0; 4. } |

\*的效率不高，所以我们使用并集选择器，罗列所有的标签（不用背，有专业的清除默认样式的样式表，今后学习）：

|  |
| --- |
| 1. body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,code,form,fieldset,legend,input,textarea,p,blockquote,th,td{ 2. margin:0; 3. padding:0 4. } |

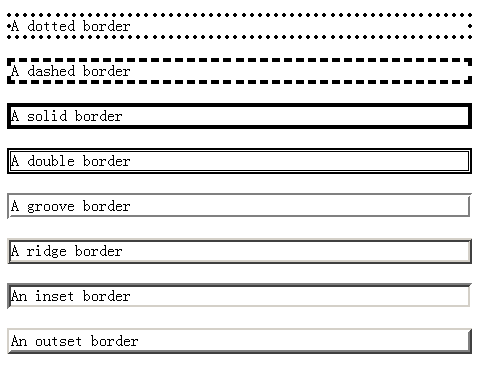
## border

就是边框。边框有三个要素：粗细、线型、颜色。

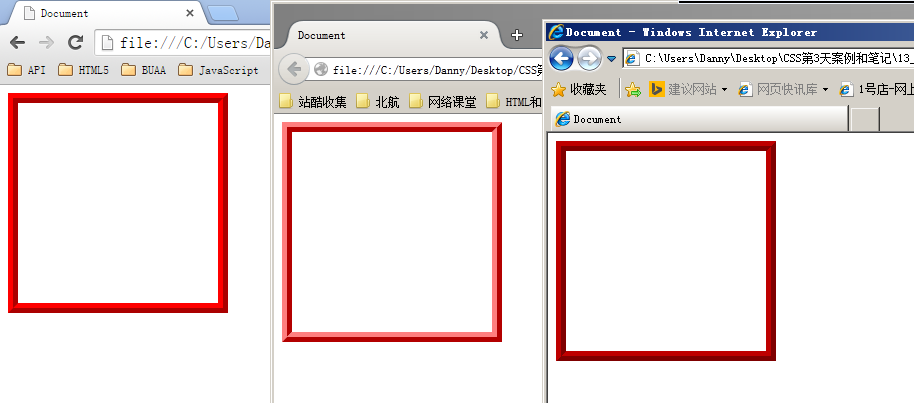
颜色如果不写，默认是黑色。另外两个属性不写，要命了，显示不出来边框。

|  |
| --- |
| 1. border: 1px dashed red; |

所有的线型：



比如，border:10px ridge red; 在chrome和firefox、IE中有细微差别：



如果公司里面的设计师，处女座的，追求极高的页面还原度，那么不能使用css来制作边框。

就要用到图片，就要切图了。所以，比较稳定的就几个：solid、dashed、dotted

border是一个大综合属性，

|  |
| --- |
| 1. border:1px solid red; |

就是把4个边框，都设置为1px宽度、线型实线、red颜色。

border属性能够被拆开，有两大种拆开的方式：

1） 按3要素:border-width、border-style、border-color

2） 按方向：border-top、border-right、border-bottom、border-left

按3要素拆开：

|  |
| --- |
| 1. **border-width**:10px; → 边框宽度 2. **border-style**:solid; → 线型 3. **border-color**:red; → 颜色。 |

等价于：

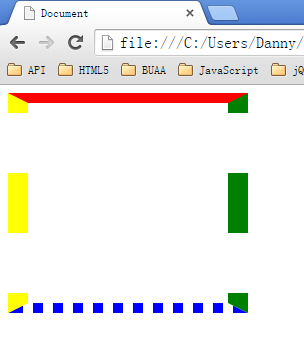
|  |
| --- |
| 1. border:10px solid red; |

现在心里要明白，**原来一个border是由三个小属性综合而成：**

**border-width border-style border-color。**

如果某一个小要素后面是空格隔开的多个值，那么就是**上右下左**的顺序：

|  |
| --- |
| 1. border-width:10px 20px; 2. border-style:**solid dashed dotted**; 3. border-color:**red green blue yellow**; |



按方向来拆

|  |
| --- |
| 1. border-top:10px solid red; 2. border-right:10px solid red; 3. border-bottom:10px solid red; 4. border-left:10px solid red; |

等价于

|  |
| --- |
| 1. border:10px solid red; |

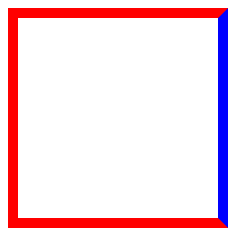
按方向还能再拆一层，就是把每个方向的，每个要素拆开，一共12条语句：

|  |
| --- |
| 1. border-top-width:10px; 2. border-top-style:solid; 3. border-top-color:red; 4. border-right-width:10px; 5. border-right-style:solid; 6. border-right-color:red; 7. border-bottom-width:10px; 8. border-bottom-style:solid; 9. border-bottom-color:red; 10. border-left-width:10px; 11. border-left-style:solid; 12. border-left-color:red; |

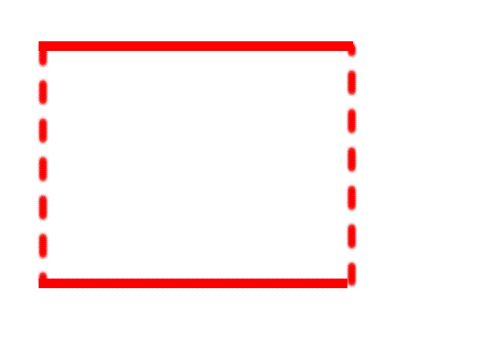
等价于

|  |
| --- |
| 1. border:10px solid red; |

工作中到底用什么？很简答：什么简单用什么？

 写法：

|  |
| --- |
| 1. border:10px solid red; 2. border-right-color:blue; |

写法：

|  |
| --- |
| 1. border:10px solid red; 2. border-style:solid dashed; |

border可以没有，

|  |
| --- |
| 1. border:none; |

某一条边没有：

|  |
| --- |
| 1. border-left: none; |

也可以调整左边边框的宽度为0：

|  |
| --- |
| 1. border-left-width: 0; |

# css布局模型

## 标准文档流

宏观的讲，我们的web页面和photoshop等设计软件有本质的区别：web页面的制作，是个“流”，必须从上而下，像“织毛衣”。而设计软件，想往哪里画个东西，都能画。

元素默认的显示方式就是标准流。

我们要看看标准流有哪些微观现象：

### 流动模型的特点

**1） 空白折叠现象：**

比如，如果我们想让img标签之间没有空隙，必须紧密连接：

|  |
| --- |
| 1. <img src="images/0.jpg" /><img src="images/1.jpg" /><img src="images/2.jpg" /> |

**2） 高矮不齐，底边对齐：**



**3） 自动换行，一行写不满，换行写。**

### 块级元素和行内元素

学习的初期，你就要知道，标准文档流等级森严。标签分为两种等级：

1） 块级元素

● 霸占一行，不能与其他任何元素并列

● 能接受宽、高

● 如果不设置宽度，那么宽度将默认变为父亲的100%。

2） 行内元素

● 与其他行内元素并排

● 不能设置宽、高。默认的宽度，就是文字的宽度。

**常用的块状元素有：**

<div>、<p>、<h1>...<h6>、<ol>、<ul>、<dl>、<table>、<address>、<blockquote> 、<form>

**常用的内联元素有：**

<a>、<span>、<br>、<i>、<em>、<strong>、<label>、<q>、<var>、<cite>、<code>

**常用的内联块状元素有：**

<img>、<input>

### 块级元素和行内元素的相互转换

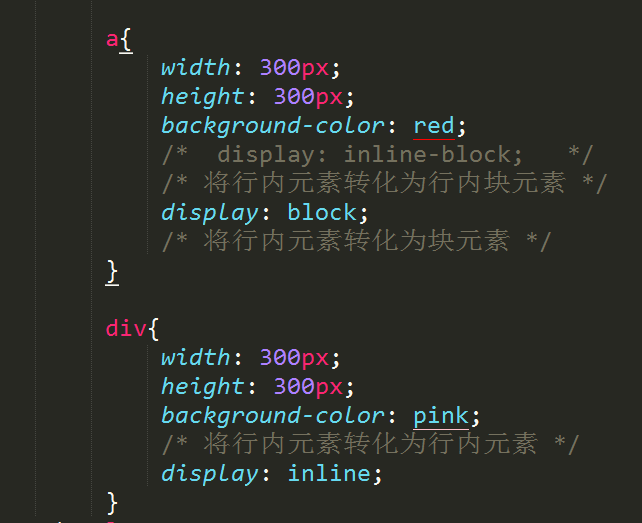
块级元素可以设置为行内元素

行内元素可以设置为块级元素

Display: inline 将元素转化为行内元素

Display：inline-block 将元素转化行内块元素

Display: block 将元素转化为块元素



|  |
| --- |
| 1. div{ 2. **display: inline;** 3. background-color: pink; 4. width: 500px; 5. height: 500px; 6. } |

display是“显示模式”的意思，用来改变元素的行内、块级性质

inline就是“行内”。

一旦，给一个标签设置

|  |
| --- |
| 1. **display: inline;** |

那么，这个标签将立即变为行内元素。此时它和一个span无异：

● 此时这个div不能设置宽度、高度；

● 此时这个div可以和别人并排了

同样的道理，

|  |
| --- |
| 1. span{ 2. **display: block;** 3. width: 200px; 4. height: 200px; 5. background-color: pink; 6. } |

“block”是“块”的意思

让标签变为块级元素。此时这个标签，和一个div无异：

● 此时这个span能够设置宽度、高度

● 此时这个span必须霸占一行了，别人无法和他并排

● 如果不设置宽度，将撑满父亲

标准流里面限制非常多，标签的性质恶心。比如，我们现在就要并排、并且就要设置宽高。

所以，移民！脱离标准流！

css中一共有三种手段，使一个元素脱离标准文档流：

1） 浮动

2） 绝对定位

3） 固定定位

## 浮动模型

特点：

☞设置了浮动的元素不占原来的位置（脱标）

☞可以让块级元素在一行上显示

☞浮动可以行内元素转化为行内块元素

模式转换的过程中，能用display就用display

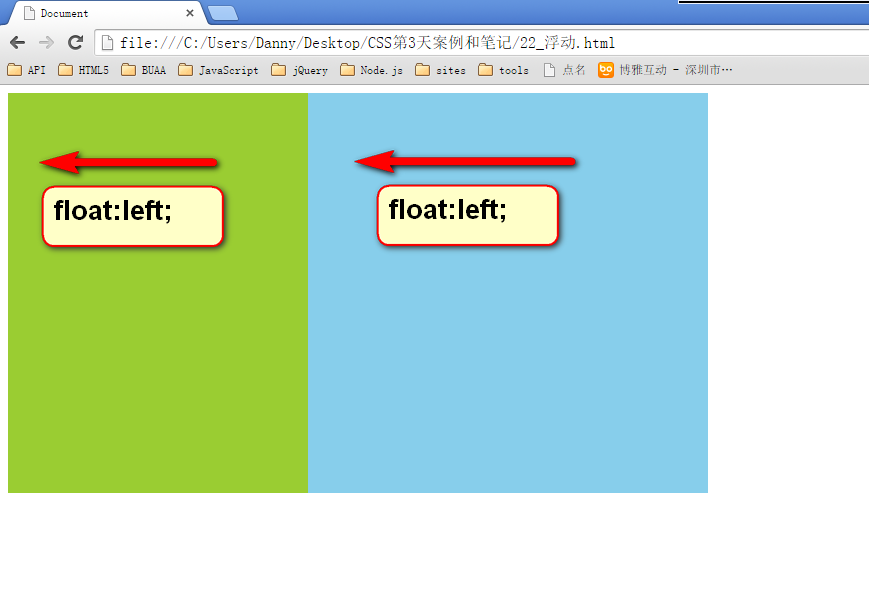
作用：

☞ 浮动用来解决文字图片环绕问题

☞ 制作导航栏

☞网页布局

浮动是css里面布局用的最多的属性。



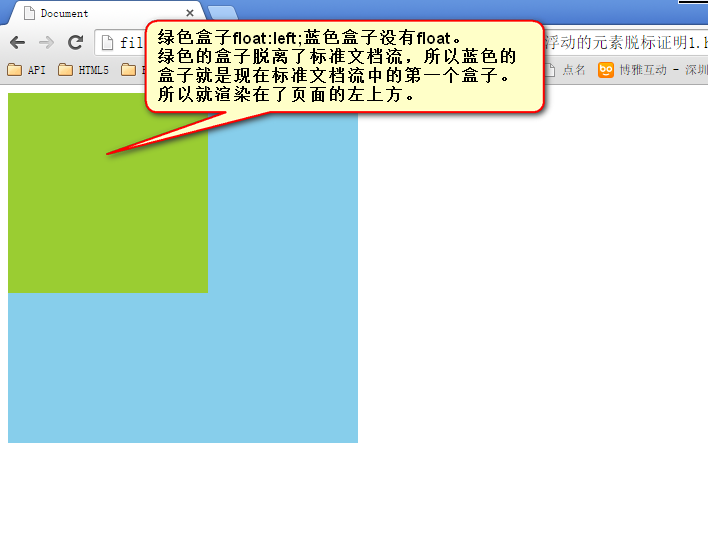
|  |
| --- |
| 1. .box1{ 2. float: left; 3. width: 300px; 4. height: 400px; 5. background-color: yellowgreen; 6. } 7. .box2{ 8. float: left; 9. width: 400px; 10. height: 400px; 11. background-color: skyblue; 12. } |

两个元素并排了，并且两个元素都能够设置宽度、高度了（这在刚才的标准流中，不能实现）。

浮动想学好，一定要知道三个性质。

### 浮动的元素脱标

证明1：



证明2：

一个span标签不需要转成块级元素，就能够设置宽度、高度了。所以能够证明一件事儿，就是所有标签已经不区分行内、块了。也就是说，**一旦一个元素浮动了，那么，将能够并排了，并且能够设置宽高了。无论它原来是个div还是个span。**

|  |
| --- |
| 1. span{ 2. **float: left;** 3. width: 200px; 4. height: 200px; 5. background-color: orange; 6. } |

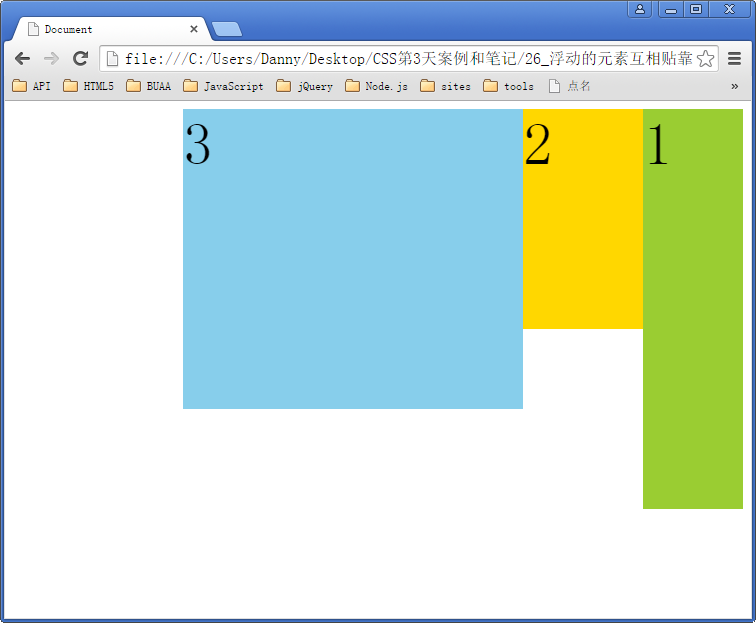
### 浮动的元素互相贴靠

如果有足够空间，那么就会靠着2哥。如果没有足够的空间，那么会靠着1号大哥。

如果没有足够的空间靠着1号大哥，自己去贴左墙。

|  |  |  |
| --- | --- | --- |
|  |  |  |

右浮动： float:right;

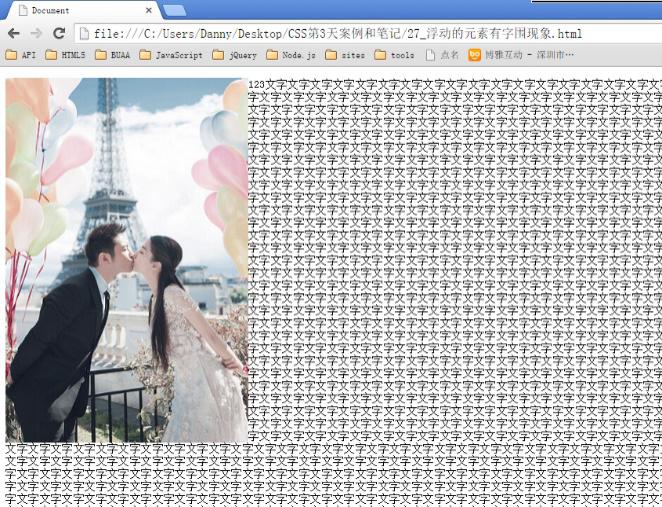


### 浮动的元素有“字围”效果

HTML：

|  |
| --- |
| 1. <div> 2. <img src="images/1.jpg" alt="" /> 3. </div> 4. <p>123文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字文字 5. </p> |

让div浮动，p不浮动：



div挡住了p，但是p中的文字不会被挡住，形成“字围”效果。

关于浮动我们要强调一点，浮动这个东西，我们在初期一定要遵循一个原则：

永远不是一个东西单独浮动，浮动都是一起浮动，要浮动，大家都浮动。

## 浮动的清除

☞清除浮动不是删除浮动

☞清除浮动指的是清除浮动的影响

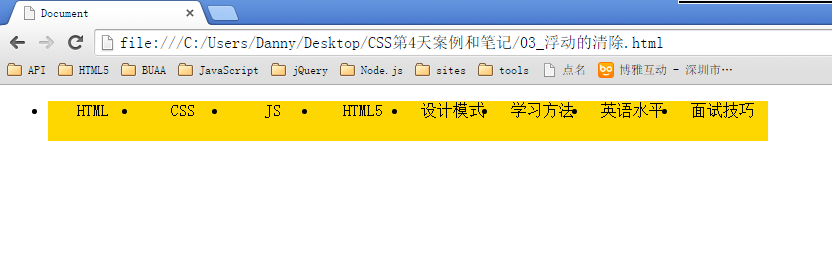
注意：

当子元素设置了浮动，父元素没有高度的时候，造成页面布局混乱。这种情况下进行清除浮动。

来看一个实验：现在有两个div，div身上没有任何属性。每个div中都有li，这些li都是浮动的。

|  |
| --- |
| 1. <div> 2. <ul> 3. <li>HTML</li> 4. <li>CSS</li> 5. <li>JS</li> 6. <li>HTML5</li> 7. <li>设计模式</li> 8. </ul> 9. </div> 10. <div> 11. <ul> 12. <li>学习方法</li> 13. <li>英语水平</li> 14. <li>面试技巧</li> 15. </ul> 16. </div> |

我们本以为这些li，会分为两排，但是，第二组中的第1个li，去贴靠第一组中的最后一个li了。

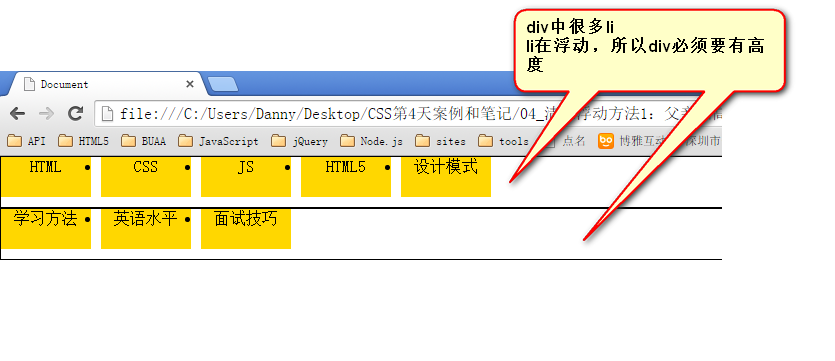


第二个div中的li，去贴第一个div中最后一个li的边了。

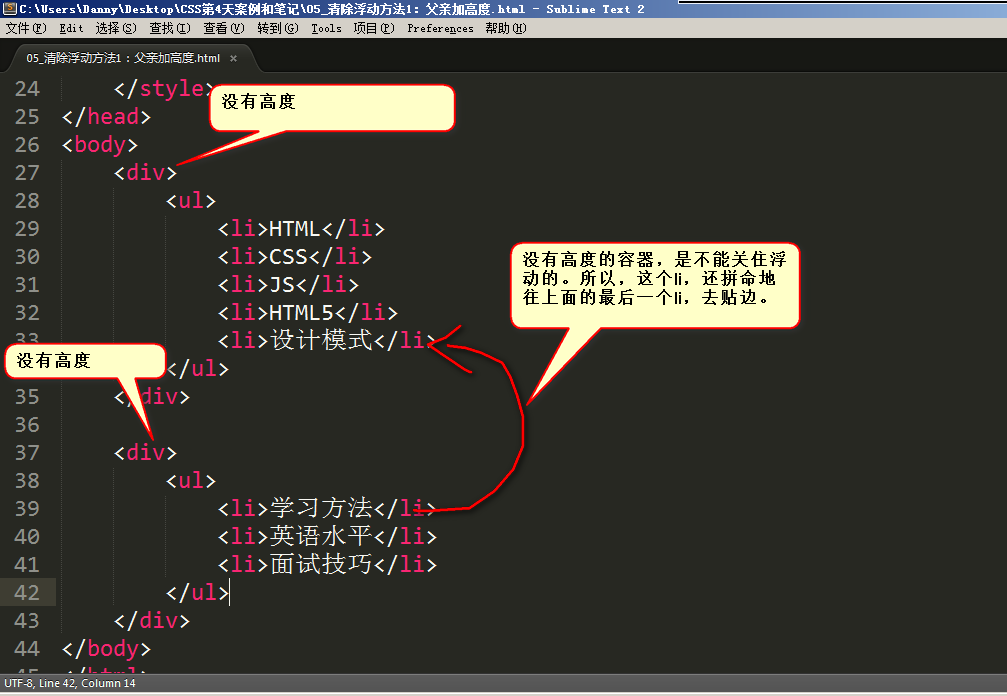
原因就是因为div没有高度，不能给自己浮动的孩子们，一个容器。

### 清除浮动方法1：给浮动的元素的祖先元素加高度。

如果一个元素要浮动，那么它的祖先元素一定要有高度。**高度的盒子，才能关住浮动。**



只要浮动在一个有高度的盒子中，那么这个浮动就不会影响后面的浮动元素。所以就是清除浮动带来的影响了。



### 清除浮动方法2：clear:both;

网页制作中，高度height很少出现。为什么？因为能被内容撑高！那也就是说，刚才我们讲解的方法1，工作中用的很少。

脑弄大开：能不能不写height，也把浮动清除了呢？也让浮动之间，互不影响呢？

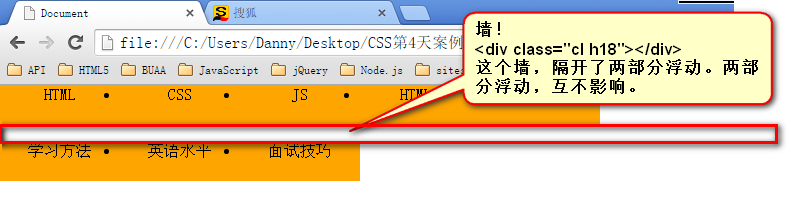
|  |
| --- |
| 1. <div> 2. <ul> 3. <li>HTML</li> 4. <li>CSS</li> 5. <li>JS</li> 6. <li>HTML5</li> 7. <li>设计模式</li> 8. </ul> 9. </div> 10. <div **class="box2"**> → 这个div写一个clear:both;属性 11. <ul> 12. <li>学习方法</li> 13. <li>英语水平</li> 14. <li>面试技巧</li> 15. </ul> 16. </div> |

|  |
| --- |
| 1. clear:both; |

clear就是清除，both指的是左浮动、右浮动都要清除。意思就是：清除别人对我的影响。

**这种方法有一个非常大的、致命的问题，margin失效了。**

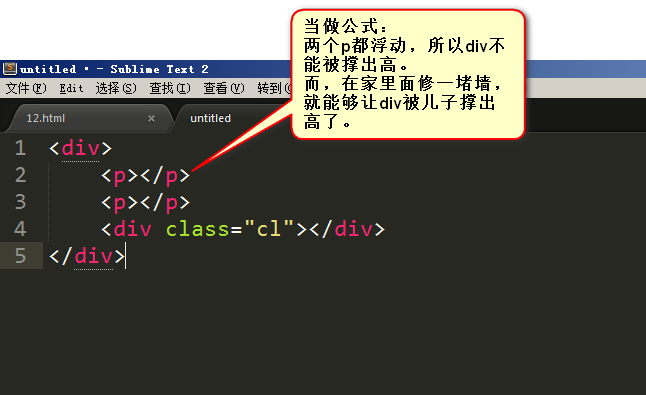
### 清除浮动方法3：隔墙法



|  |
| --- |
| 1. <div class="box1"> 2. <ul> 3. <li>HTML</li> 4. <li>CSS</li> 5. <li>JS</li> 6. <li>HTML5</li> 7. <li>设计模式</li> 8. </ul> 9. </div> 11. **<div class="cl h16"></div>** 12. <div class="box2"> 13. <ul> 14. <li>学习方法</li> 15. <li>英语水平</li> 16. <li>面试技巧</li> 17. </ul> 18. </div> |

|  |
| --- |
| 1. .cl{ 2. clear: both; 3. } 4. .h16{ 5. height: 16px; 6. } |

近些年，有演化出了“内墙法”：



### 清除浮动方法4：overflow:hidden;

overflow就是“溢出”的意思， hidden就是“隐藏”的意思。

|  |
| --- |
| 1. overflow:hidden; |

表示“溢出隐藏”。所有溢出边框的内容，都要隐藏掉。

|  |  |
| --- | --- |
| 内容太多，溢出了盒子： | overflow:hidden; 溢出盒子边框的内容，隐藏了。 |

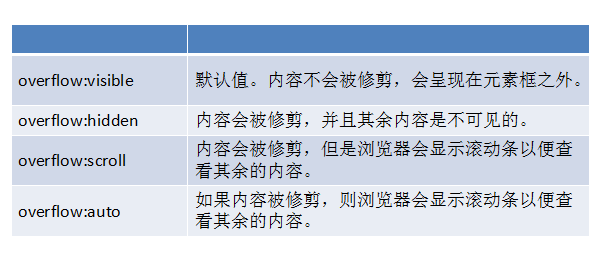
本意就是清除溢出到盒子外面的文字。但是，前端开发工程师又发现了，它能做偏方。

一个父亲不能被自己浮动的儿子，撑出高度。但是，只要给父亲加上overflow:hidden; 那么，父亲就能被儿子撑出高了。这是一个偏方。

|  |
| --- |
| 1. div{ 2. width: 400px; 3. border: 10px solid black; 4. **overflow: hidden;** 5. } |

|  |  |
| --- | --- |
|  |  |





## 清除浮动总结与案例

总结一下：

1） 加高法：

浮动的元素，只能被有高度的盒子关住。 也就是说，如果盒子内部有浮动，这个盒子有高，那么妥妥的，浮动不会互相影响。但是，工作上，我们绝对不会给所有的盒子加高度，这是因为麻烦，并且不能适应页面的快速变化。

|  |
| --- |
| 1. <div> **→ 设置height** 2. <p></p> 3. <p></p> 4. <p></p> 5. </div> 6. <div> **→ 设置height** 7. <p></p> 8. <p></p> 9. <p></p> 10. </div> |

2) clear:both;法

最简单的清除浮动的方法，就是给盒子增加clear:both；表示自己的内部元素，不受其他盒子的影响。

|  |
| --- |
| 1. <div> 2. <p></p> 3. <p></p> 4. <p></p> 5. </div> 6. <div> **→ clear:both;** 7. <p></p> 8. <p></p> 9. <p></p> 10. </div> |

浮动确实被清除了，不会互相影响了。但是有一个问题，就是margin失效。两个div之间，没有任何的间隙了。

3）隔墙法：

在两部分浮动元素中间，建一个墙。隔开两部分浮动，让后面的浮动元素，不去追前面的浮动元素。

墙用自己的身体当做了间隙。

|  |
| --- |
| 1. <div> 2. <p></p> 3. <p></p> 4. <p></p> 5. </div> 6. **<div class="cl h10"></div>** 7. <div> 8. <p></p> 9. <p></p> 10. <p></p> 11. </div> |

我们发现，隔墙法好用，但是第一个div，还是没有高度。如果我们现在想让第一个div，自动的根据自己的儿子，撑出高度，我们就要想一些“小伎俩”，“奇淫技巧”。

内墙法：

|  |
| --- |
| 1. <div> 2. <p></p> 3. <p></p> 4. <p></p> 5. **<div class="cl h10"></div>** 6. </div> 7. <div> 8. <p></p> 9. <p></p> 10. <p></p> 11. </div> |

内墙法的优点就是，不仅仅能够让后部分的p不去追前部分的p了，并且能把第一个div撑出高度。这样，这个div的背景、边框就能够根据p的高度来撑开了。

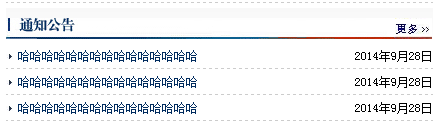
4）overflow:hidden;

这个属性的本意，就是将所有溢出盒子的内容，隐藏掉。但是，我们发现这个东西能够用于浮动的清除。

我们知道，一个父亲，不能被自己浮动的儿子撑出高度，但是，如果这个父亲加上了overflow:hidden；那么这个父亲就能够被浮动的儿子撑出高度了。这个现象，不能解释，就是浏览器的小偏方。

并且,overflow:hidden;能够让margin生效。

清除浮动的案例：



实践中，遇见了清除浮动的问题：



## 浏览器兼容问题

**上述知识点遇见的浏览器兼容问题**

第一，IE6，不支持小于12px的盒子，任何小于12px的盒子，在IE6中看都大

解决办法很简单，就是将盒子的字号，设置小（小于盒子的高），比如0px。

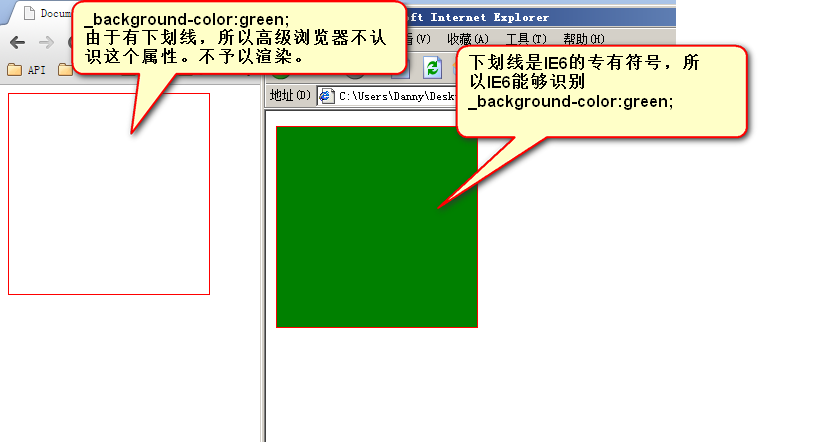
|  |
| --- |
| 1. height: 4px; 2. \_font-size: 0px; |

我们现在介绍一下浏览器hack。hack就是“黑客”，就是使用浏览器提供的后门，针对某一种浏览器做兼容。

**IE6留了一个后门，就是只要给css属性之前，加上下划线，这个属性就是IE6认识的专有属性。**

比如：

|  |
| --- |
| 1. \_background-color: green; |



解决微型盒子，正确写法：

|  |
| --- |
| 1. height: 10px; 2. **\_font-size:0;** |

第二，IE6不支持用overflow:hidden;来清除浮动的

解决办法，以毒攻毒。追加一条

|  |
| --- |
| 1. \_zoom:1; |

完整写法：

|  |
| --- |
| 1. overflow: hidden; 2. **\_zoom:1;** |

实际上，\_zoom:1;能够触发浏览器hasLayout机制。这个机制，不要深究了，因为就IE6有。我们只需要让IE6好用，具体的实现机制，有兴趣的同学，自行百度。

强调一点， overflow:hidden;的本意，就是溢出盒子的border的东西隐藏，这个功能是IE6兼容的。不兼容的是overflow:hidden;清除浮动的时候。

我们刚才学习了两个IE6的兼容问题，这两个IE6的兼容问题，都是通过多写一条hack来解决的。

这个我们称为伴生属性。

|  |
| --- |
| 1. height:6px; 2. **\_font-size:0;** |

|  |
| --- |
| 1. overflow:hidden; 2. **\_zoom:1;** |

## 层模型

层模型有三种形式：

1、绝对定位(position: absolute)

2、相对定位(position: relative)

3、固定定位(position: fixed)

### 认识相对定位

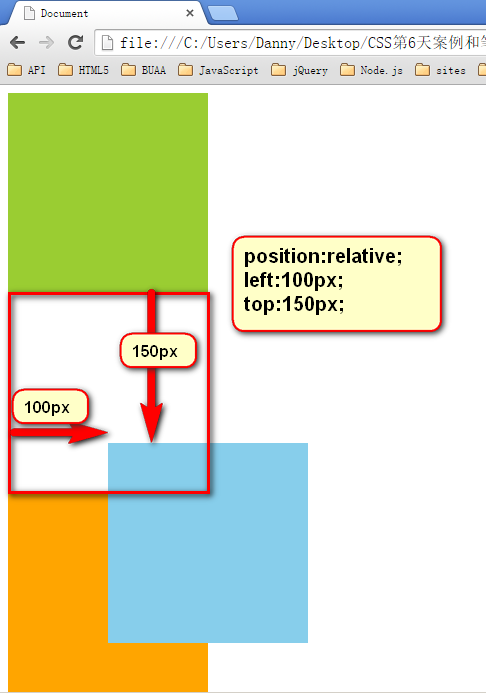
☞元素设置了相对定位后占原来的位置

☞设置相对定位以自己的位置为参照设置位置

☞相对定位不能进行元素的模式转换

☞子绝父相（子元素设置绝对定位，父元素设置相对定位）

相对定位，就是微调元素位置的。让元素**相对自己原来的位置，进行位置调整**。



也就是说，如果一个盒子想进行位置调整，那么就要使用相对定位

|  |
| --- |
| 1. position:relative; → 必须先声明，自己要相对定位了， 2. left:100px; → 然后进行调整。 3. top:150px; → 然后进行调整。 |

### **不脱标，老家留坑，形影分离**

相对定位不脱标，真实位置是在老家，只不过影子出去了，可以到处飘。



### 相对定位用途

相对定位有坑，所以一般不用于做“压盖”效果。页面中，效果极小。就两个作用：

1） **微调**元素

2） 做绝对定位的参考，子绝父相（讲绝对定位的时候说）

### 相对定位的定位值

可以用left、right来描述盒子右、左的移动；

可以用top、bottom来描述盒子的下、上的移动。

↘：

|  |
| --- |
| 1. position: relative; 2. top: 10px; 3. left: 40px; |

↙：

|  |
| --- |
| 1. position: relative; 2. **right: 100px; → 往左边移动** 3. top: 100px; |

↖：

|  |
| --- |
| 1. position: relative; 2. right: 100px; 3. **bottom: 100px; → 移动方向是向上。** |

↗：

|  |
| --- |
| 1. position: relative; 2. **top: -200px; → 负数就是相反的方向，如果是正，就是下边，如果是负数就是上边** 3. **right: -200px;** |

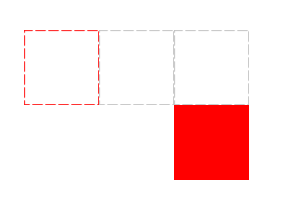
↗：

|  |
| --- |
| 1. position: relative; 2. **right: -300px;** 3. bottom: 300px; |

完全等价于：

|  |
| --- |
| 1. position: relative; 2. **left: 300px;** 3. bottom: 300px; |

如图，有几种相对定位的移动方法？



方法1：

|  |
| --- |
| 1. position:relative; 2. top:100px; 3. left:200px; |

方法2：

|  |
| --- |
| 1. position:relative; 2. bottom:-100px; 3. right:-200px; |

方法3：

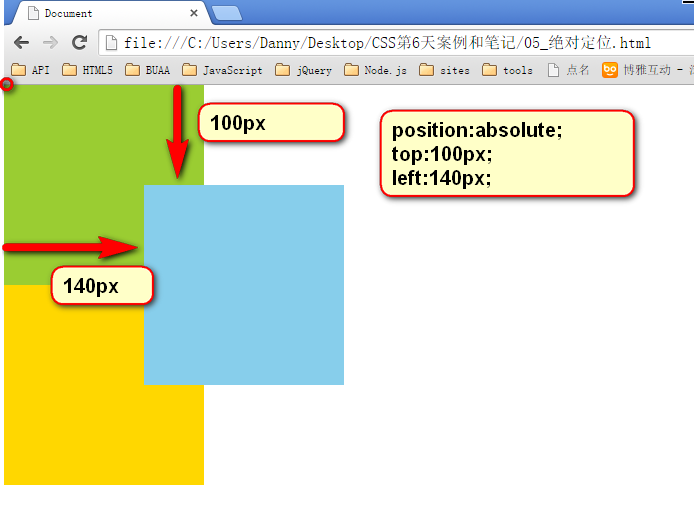
|  |
| --- |
| 1. position:relative; 2. top:100px; 3. right:-200px; |

方法4：

|  |
| --- |
| 1. position:relative; 2. bottom:-100px; 3. left:200px; |

## 绝对定**位**

绝对定位比相对定位更灵活。



### 绝对定位脱标

特点：

☞当给一个单独的元素设置绝对定位，以浏览器左上角（body）为基准设置定位的。

☞当盒子发生嵌套关系的时候，如果父盒子没有设置定位，子盒子设置定位以浏览器左上角为基准设置定位。

☞当盒子发生嵌套关系的时候，如果父盒子设置定位，子盒子设置定位父盒子左上角为基准设置定位。

☞给盒子设置了绝对定位，该盒子不占位置（脱标）

☞给行内元素设置绝对定位后，该元素转化为了行内块元素

注意：元素设置了绝对定位后，通过具体的方位名称可以随便设置元素的位置。

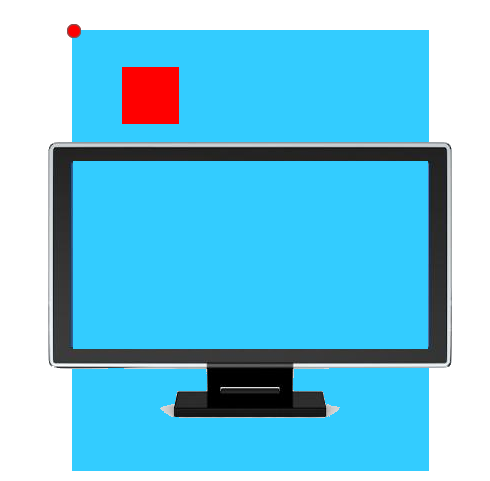
绝对定位的盒子，是脱离标准文档流的。所以，所有的标准文档流的性质，绝对定位之后都不遵守了。

绝对定位之后，标签就不区分所谓的行内元素、块级元素了，不需要display:block;就可以设置宽、高了：

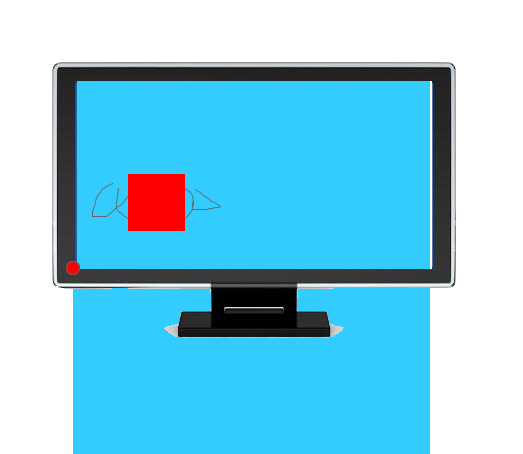
|  |
| --- |
| 1. span{ 2. **position: absolute;** 3. top: 100px; 4. left: 100px; 5. width: 100px; 6. height: 100px; 7. background-color: pink; 8. } |

### 参考点

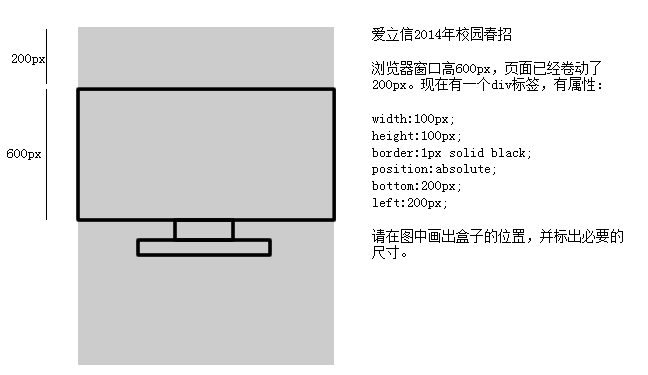
**绝对定位的参考点，如果用top描述，那么定位参考点就是页面的左上角，而不是浏览器的左上角：**



**如果用bottom描述，那么就是浏览器首屏窗口尺寸，对应的页面的左下角**：

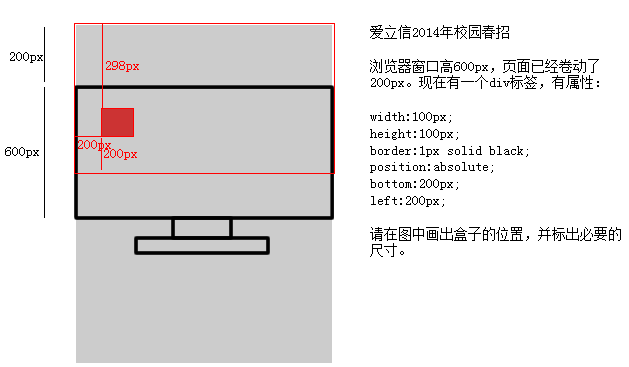


面试题：



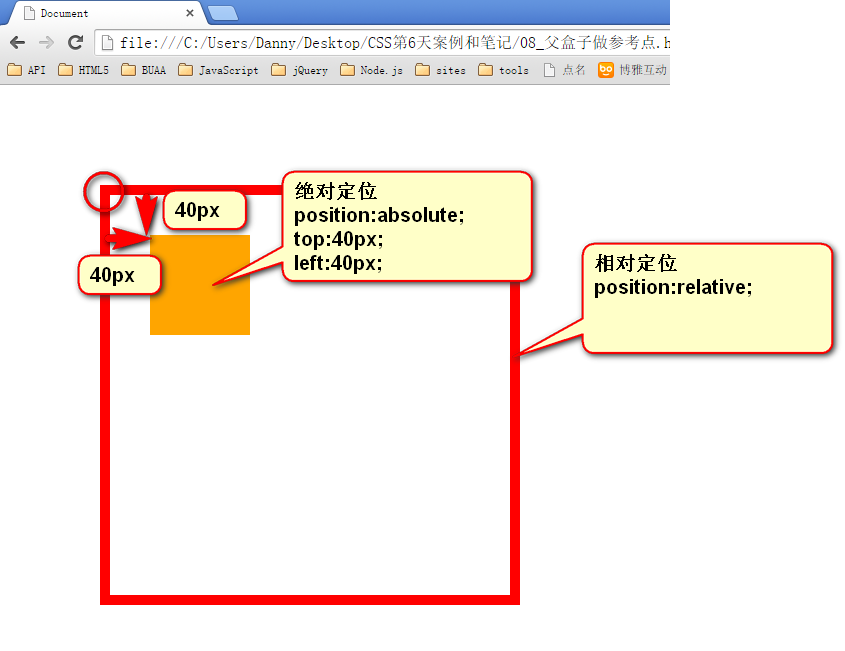
答案：

用bottom的定位的时候，参考的是浏览器首屏大小对应的页面左下角。



## **以盒子为参考点**

一个绝对定位的元素，如果父辈元素中出现了也定位了的元素，那么将以父辈这个元素，为参考点。



● 要听最近的已经定位的祖先元素的，不一定是父亲，可能是爷爷：

|  |
| --- |
| 1. <div class="box1"> → 相对定位 2. <div class="box2"> → 没有定位 3. <p></p> → 绝对定位，将以box1为参考，因为box2没有定位，box1就是最近的父辈元素 4. </div> 5. </div> |

|  |
| --- |
| 1. <div class="box1"> → 相对定位 2. <div class="box2"> → 相对定位 3. <p></p> → 绝对定位，将以box2为参考，因为box2是自己最近的父辈元素 4. </div> 5. </div> |

● 不一定是相对定位，任何定位，都可以作为参考点

|  |
| --- |
| 1. <div> → 绝对定位 2. <p></p> → 绝对定位，将以div作为参考点。因为父亲定位了。 3. </div> |

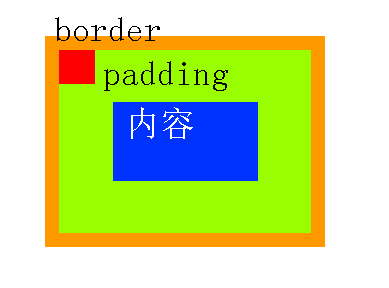
子绝父绝、子绝父相、子绝父固，都是可以给儿子定位的。但是，工程上子绝、父绝，没有一个盒子在标准流里面了，所以页面就不稳固，没有任何实战用途。工程上，“子绝父相”有意义，父亲没有脱标，儿子脱标在父亲的范围里面移动。

|  |
| --- |
| 1. <div class=”box1”> → 绝对定位 2. <div class=”box2”> → 相对定位 3. <div class=”box3”> → 没有定位 4. <p></p> → 绝对定位，以box2为参考定位。 5. </div> 6. </div> 7. </div> |

● 绝对定位的儿子，无视参考的那个盒子的padding。

下图中，绿色部分是div的padding，蓝色部分是div的内容区域。那么此时，div相对定位，p绝对定位。

p将无视父亲的padding，在border内侧为参考点，进行定位：



## **绝对定位的盒子居中**

绝对定位之后，所有标准流的规则，都不适用了。所以margin:0 auto;失效。

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| 1. width: 600px; 2. height: 60px;   **position: absolute;**   1. **left: 50%;** 2. top: 0; 3. **margin-left: -300px; → 宽度的一半** |

非常简单，当做公式记忆下来。就是left:50%; margin-left:负的宽度的一半。

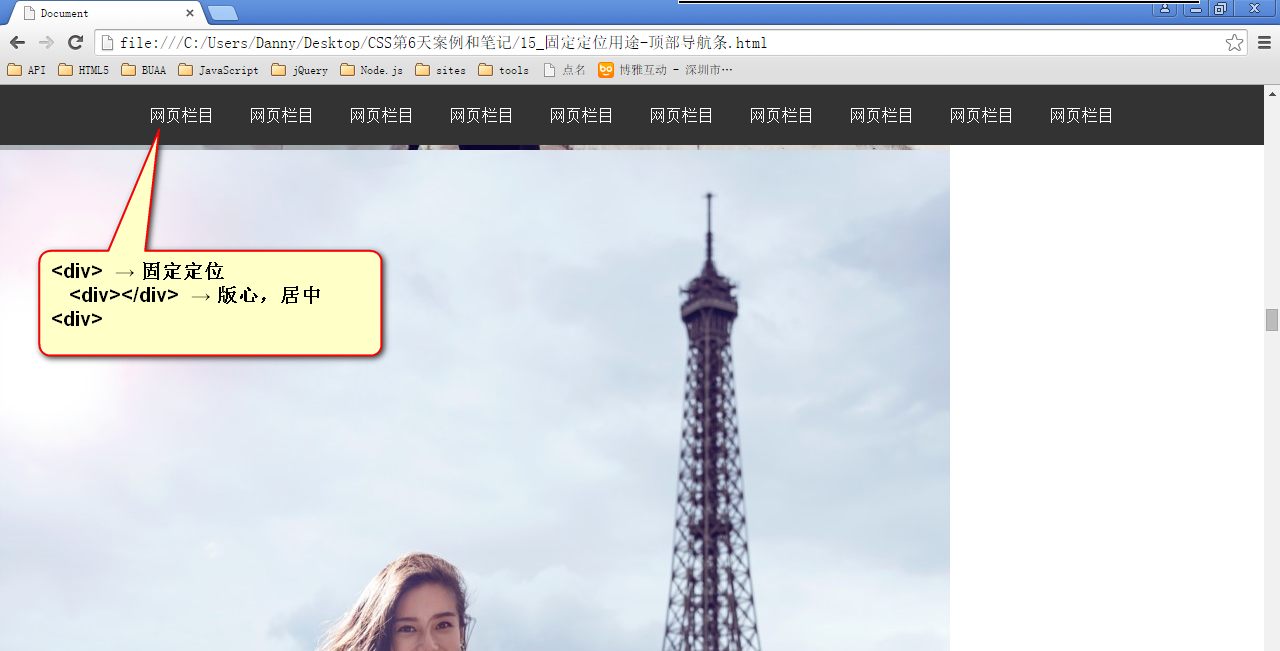
## 固定定位

☞固定定位不占位置（脱标）

☞将行内元素转化为行内块元素

固定定位，就是相对浏览器窗口定位。页面如何滚动，这个盒子显示的位置不变。

固定定位脱标！



**IE6不兼容。**

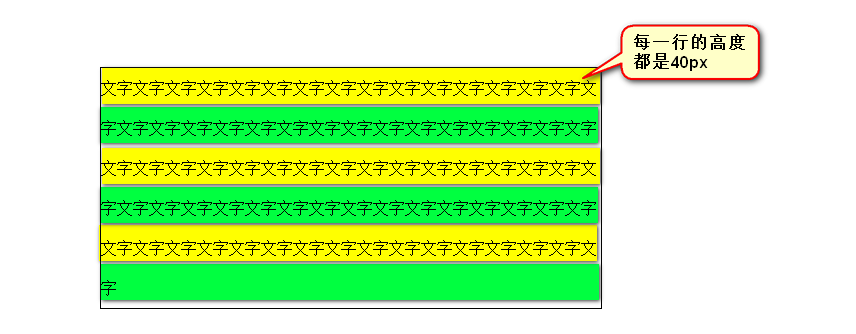
# 文本样式

## 行高和字号

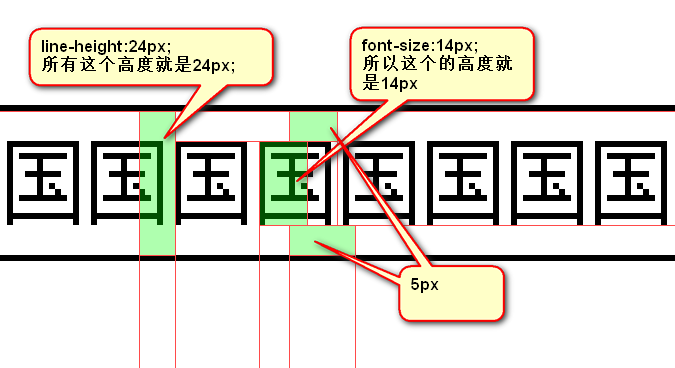
## 行高

CSS中，所有的行，都有行高。盒模型的padding，绝对不是直接作用在文字上的，而是作用在“行”上的。

|  |
| --- |
| 1. line-height: 40px; |



文字，是在自己的行里面居中的。比如，现在文字字号14px，行高是24px。那么：



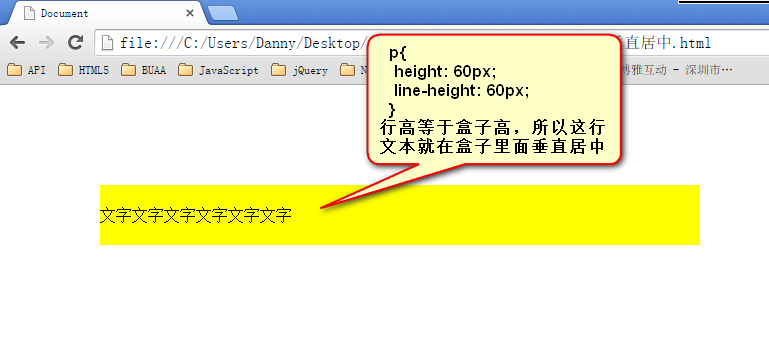
为了严格保证字在行里面居中，我们的工程师有一个约定： **行高、字号，一般都是偶数。**这样，它们的差，就是偶数，就能够被2整除。

## 单行文本垂直居中

文本在行里面居中

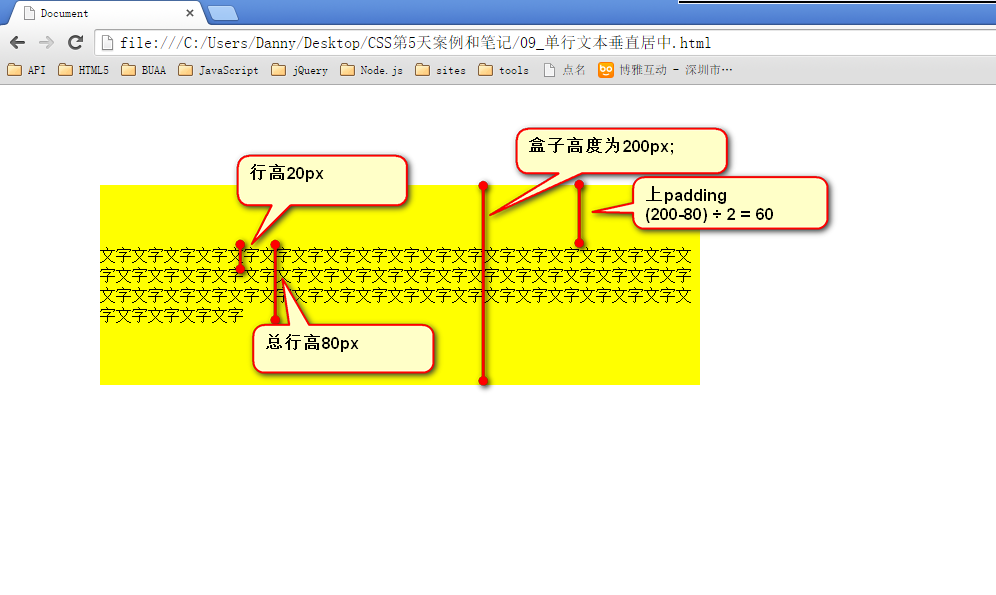
公式：

|  |
| --- |
| 1. 行高:盒子高; |



需要注意的是，这个小技巧，行高=盒子高。 **只适用于单行文本垂直居中！！不适用于多行**。

**如果想让多行文本垂直居中，需要设置盒子的padding**：



文本水平对齐 text-align:left|center|right

文本缩进 text-indent:value

例如:首行缩进 ：text-indent:2em;

## font属性

**● 使用font属性，能够将字号、行高、字体，能够一起设置**。

|  |
| --- |
| 1. **font: 14px/24px “宋体”;** |

等价于三行语句：

|  |
| --- |
| 1. font-size:14px; 2. line-height:24px; 3. font-family:"宋体"; |

font-family就是“字体”。family是“家庭”、“伐木累”的意思。

**● 网页中不是所有字体都能用哦，因为这个字体要看用户的电脑里面装没装**，比如你设置：

|  |
| --- |
| 1. font-family: "华文彩云"; |

如果用户电脑里面没有这个字体，那么就会变成宋体。

**页面中，中文我们只使用： 微软雅黑、宋体、黑体。 如果页面中，需要其他的字体，那么需要切图。**

**英语：Arial 、 Times New Roman**

**● 为了防止用户电脑里面，没有微软雅黑这个字体。就要用英语的逗号，隔开备选字体**，就是说如果用户电脑里面，没有安装微软雅黑字体，那么就是宋体：

|  |
| --- |
| 1. font-family: "微软雅黑","宋体"; |

备选字体可以有无数个，用逗号隔开。

**● 我们要将英语字体，放在最前面，这样所有的中文，就不能匹配英语字体，就自动的变为后面的中文字体**：

|  |
| --- |
| 1. font-family: "Times New Roman","微软雅黑","宋体"; |

**● 所有的中文字体，都有英语别名，我们也要知道**：

微软雅黑的英语别名：

|  |
| --- |
| 1. font-family: "Microsoft YaHei"; |

宋体的英语别名：

|  |
| --- |
| 1. font-family: "SimSun"; |

font属性能够将font-size、line-height、font-family合三为一：

|  |
| --- |
| 1. font:12px/30px "Times New Roman"**,**"Microsoft YaHei"**,**"SimSun"; |

**● 行高可以用百分比，表示字号的百分之多少**。一般来说，都是大于100%的，因为行高一定要大于字号。

|  |
| --- |
| 1. font:12px/**200%** “宋体” |

等价于

|  |
| --- |
| 1. font:12px/**24px** “宋体”; |

反过来，比如：

|  |
| --- |
| 1. font:16px/**48px** “宋体”; |

等价于

|  |
| --- |
| 1. font:16px/**300%** “宋体” |

### 字体相关属性

Font-size: 设置文字大小

Font-family: 设置文字字体

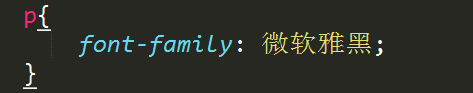
Font-weight: 设置文字加粗

Font-style: 设置文字斜体

Line-height: 行高

### Font-family

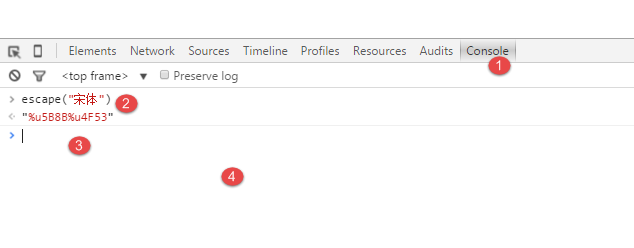
☞ 直接设置汉字

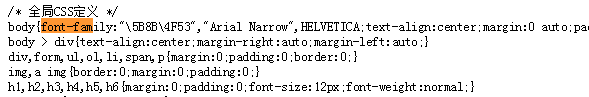


☞ 设置对应字体的英文

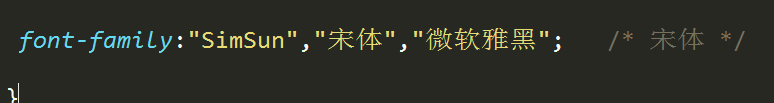


☞可以设置unicode编码





Font-family可以设置多个值；



### Font-weight

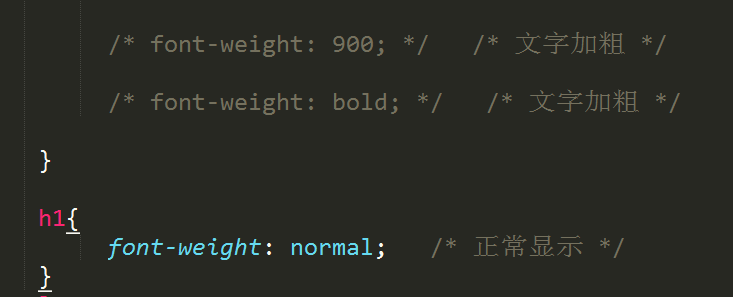
取值方式：

☞ 直接设置数字 100-900

☞ bold (字体加粗) 效果与 700-900

☞ normal(文字字体正常显示)

注意：文字加粗显示，推荐使用 font-weight:700;



### Font-styl e(斜体)

☞ italic 文字斜体显示

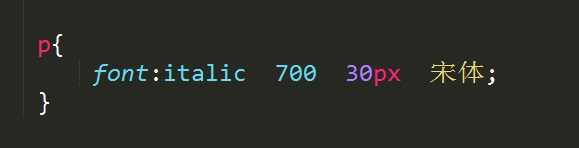
☞ normal 文字正常显示

### Font属性联写

用法：

font: font-style font-weight font-size/line-height

font-family



注意：

☞ font属性联写必须有 font-size 和font-family(其他属性可以不写)

☞ font-size 和font-family的顺序不能换

# **background系列属性**

## background-color属性

背景颜色属性。

css2.1中，颜色的表示方法有哪些？一共有三种：单词、rgb表示法、十六进制表示法

### 用英语单词来表示

能够用英语单词来表述的颜色，都是简单颜色。

红色：

|  |
| --- |
| 1. background-color: red; |

### 用rgb方法来表示

红色：

|  |
| --- |
| 1. background-color: **rgb(255,0,0)**; |

rgb表示三原色“红”red、“绿”green、“蓝”blue。光学显示器，每个像素都是由三原色的发光原件组成的，靠明亮度不同调成不同的颜色的。

**用逗号隔开，r、g、b的值，每个值的取值范围0~255，一共256个值。**

如果此项的值，是255，那么就说明是纯色：

绿色：

|  |
| --- |
| 1. background-color: rgb(0,255,0); |

蓝色：

|  |
| --- |
| 1. background-color: rgb(0,0,255); |

黑色：

|  |
| --- |
| 1. background-color: rgb(0,0,0); |

光学显示器，每个元件都不发光，黑色的。

白色：

|  |
| --- |
| 1. background-color: rgb(255,255,255); |

颜色可以叠加，比如黄色就是红色和绿色的叠加：

|  |
| --- |
| 1. background-color: rgb(255,255,0); |

再比如：

|  |
| --- |
| 1. background-color: rgb(111,222,123); |

就是红、绿、蓝三种颜色的不同比例叠加。

### 十六进制表示法

红色：

|  |
| --- |
| 1. background-color: **#ff0000**; |

所有用#开头的值，都是16进制的。

#ff0000

16进制表示法，也是两位两位看，看r、g、b，但是没有逗号隔开。

ff就是10进制的255 ，00 就是10进制的0，00就是10进制的0。所以等价于rgb(255,0,0);

怎么换算的？我们介绍一下

我们现在看一下10进制中的基本数字（一共10个）:

0、1、2、3、4、5、6、7、8、9

16进制中的基本数字（一共16个）:

0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f

16进制对应表：

十进制数 十六进制数

0 0

1 1

2 2

3 3

……

10 a

11 b

12 c

13 d

14 e

15 f

16 10

17 11

18 12

19 13

……

43 2b

……

255 ff

十六进制中，13 这个数字表示什么？

**表示1个16和3个1。 那就是19**。 这就是**位权**的概念，开头这位表示多少个16，末尾这位表示多少个1。

小练习：

16进制中28等于10进制多少？

答：2\*16+8 = 40。

16进制中的2b等于10进制多少？

答：2\*16+11 = 43。

16进制中的af等于10进制多少？

答：10 \* 16 + 15 = 175

16进制中的ff等于10进制多少？

答：15\*16 + 15 = 255

所以，#ff0000就等于rgb(255,0,0)

|  |
| --- |
| 1. background-color: #**123456**; |

等价于：

|  |
| --- |
| 1. background-color: rgb(**18**,**52**,**86**); |

所以，**任何一种十六进制表示法，都能够换算成为rgb表示法。也就是说，两个表示法的颜色数量，一样**。

**十六进制可以简化为3位，所有#aabbcc的形式，能够简化为#abc;**

比如：

|  |
| --- |
| 1. background-color:#ff0000; |

等价于

|  |
| --- |
| 1. background-color:#f00; |

比如：

|  |
| --- |
| 1. background-color:#112233; |

等价于

|  |
| --- |
| 1. background-color:#123; |

只能上面的方法简化，比如

|  |
| --- |
| 1. background-color:#222333; |

无法简化！

再比如

|  |
| --- |
| 1. background-color:#123123; |

无法简化！

要记住：

|  |
| --- |
| 1. #000 黑 2. #fff 白 3. #f00 红 4. #333 灰 5. #222 深灰 6. #ccc 浅灰 |

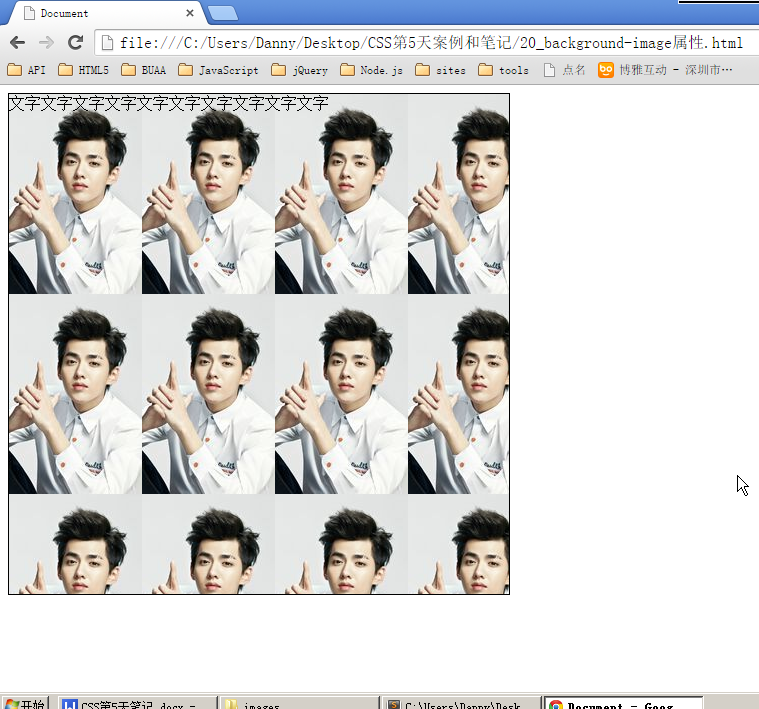
## background-image

用于给盒子加上背景图片：

|  |
| --- |
| 1. background-image:**url(images/wuyifan.jpg)**; |

url()表示网址，uniform resouces locator 同意资源定位符

images/wuyifan.jpg 就是相对路径。



背景天生是会被平铺满的。

padding的区域有背景图。

## background-repeat属性

设置背景图是否重复的，重复方式的。

repeat表示“重复”。

|  |  |  |  |
| --- | --- | --- | --- |
| 默认 | background-repeat:no-repeat; | background-repeat:repeat-x; | background-repeat:repeat-y; |

也就是说，background-repeat属性，有三种值：

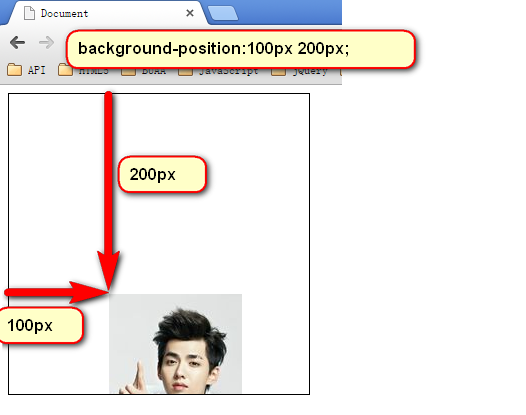
|  |
| --- |
| 1. background-repeat:no-repeat; 不重复 2. background-repeat:repeat-x; 横向重复 3. background-repeat:repeat-y; 纵向重复 |

## background-position属性

设置具体值： left| right| top| bottom| cneter

### 属性的意思

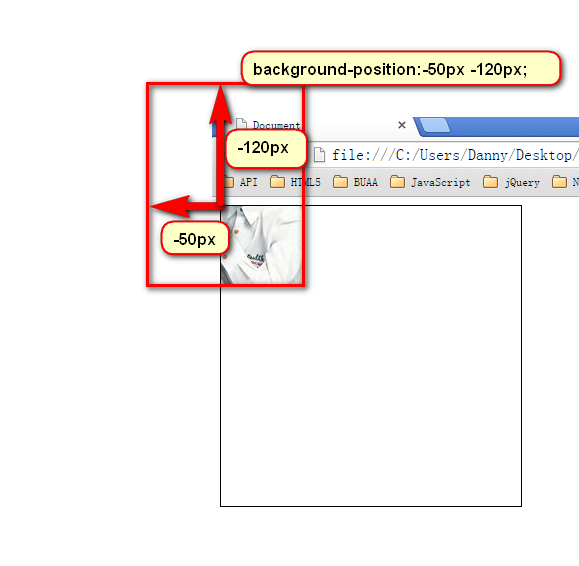
背景定位属性，是最难的属性。一定要好好学。



position就是“位置”的意思。background-position就是背景定位属性。

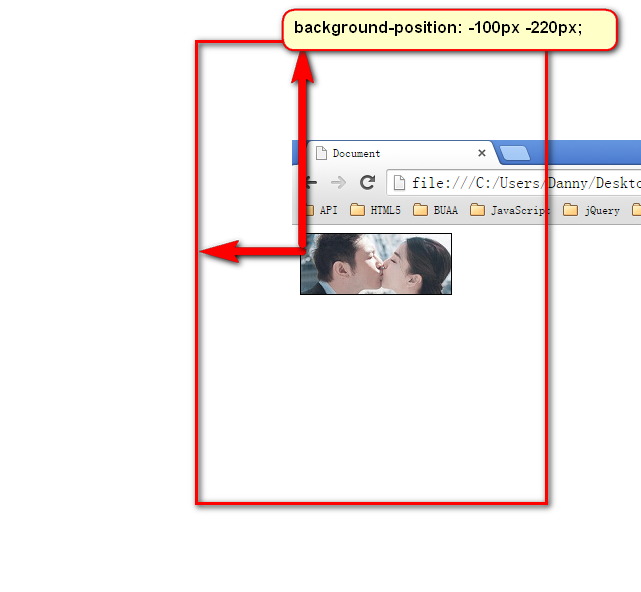
|  |
| --- |
| 1. background-position:向右移动量 向下移动量; |

定位属性可以是负数：



### css精灵

原理：

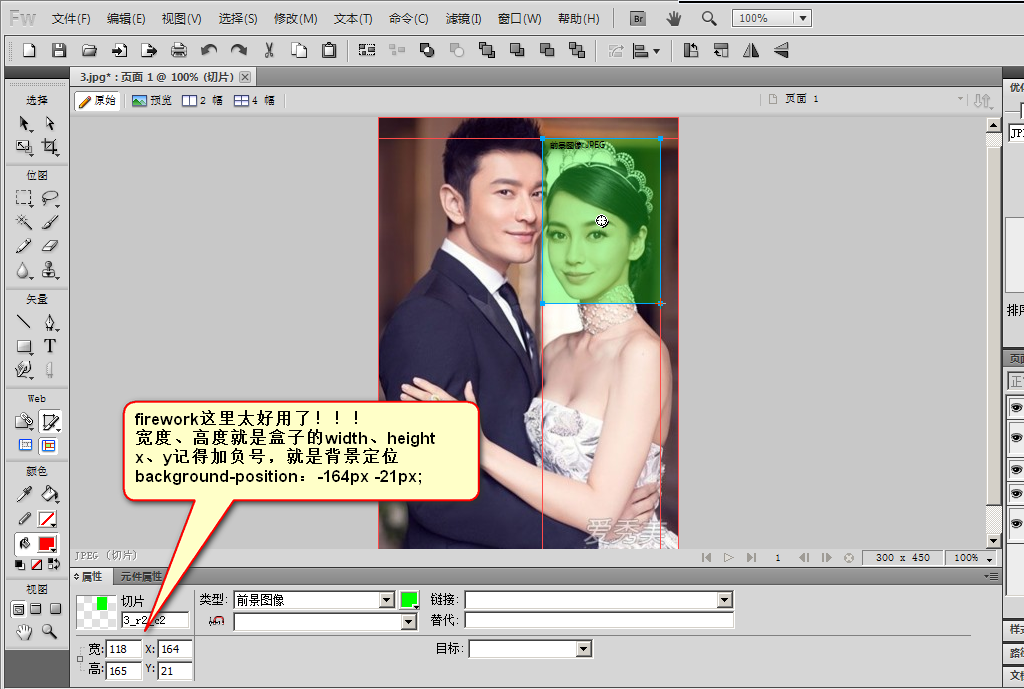


“css精灵”，英语css sprite，所以也叫做“css雪碧”技术。是一种CSS图像合并技术，该方法是将小图标和背景图像合并到一张图片上，然后利用css的背景定位来显示需要显示的图片部分。

css精灵有什么优点，就是减少了http请求。比如4张小图片，原本需要4个http请求。但是用了css精灵，小图片变为了一张图，http请求只有1个了。

淘宝网的精灵图：

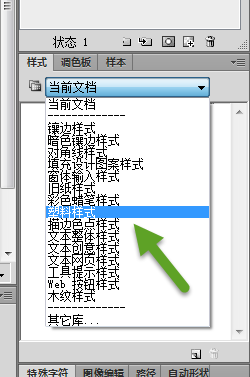
 用fireworks精确控制精灵：

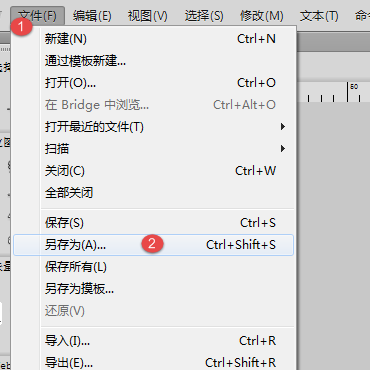


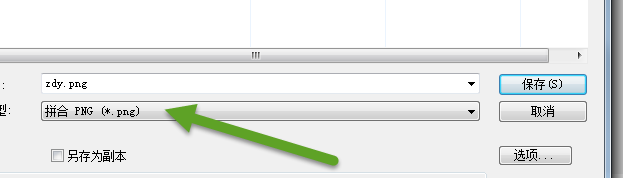
### 制作精灵图步骤

☞选择透明文档



☞

☞

☞

### 15.4.3 用单词描述

|  |
| --- |
| 1. background-position: 描述左右的词儿 描述上下的词儿; |

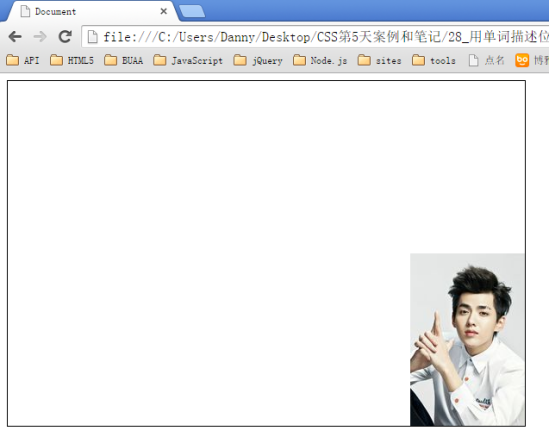
描述左右的词儿： left、 center、right

描述上下的词儿： top 、center、bottom

所以：

|  |
| --- |
| 1. background-position: right bottom; |

右下角：

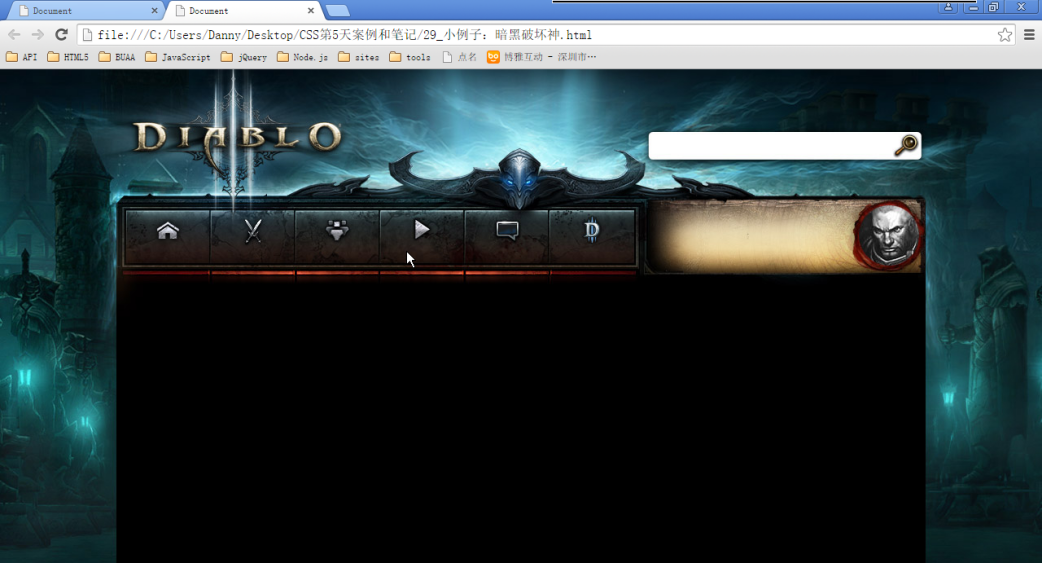


左下角：

|  |
| --- |
| 1. background-position: left bottom; |

用图：

1） 大背景图居中



2） 通栏banner



## background-attachment

scroll (默认值)滚动 fixed图片固定

背景是否固定。

|  |
| --- |
| 1. background-attachment:fixed; |

背景就会被固定住，不会被滚动条滚走。

## background综合属性

background属性和border一样，是一个综合属性：

|  |
| --- |
| 1. background:red url(1.jpg) no-repeat 100px 100px fixed; |

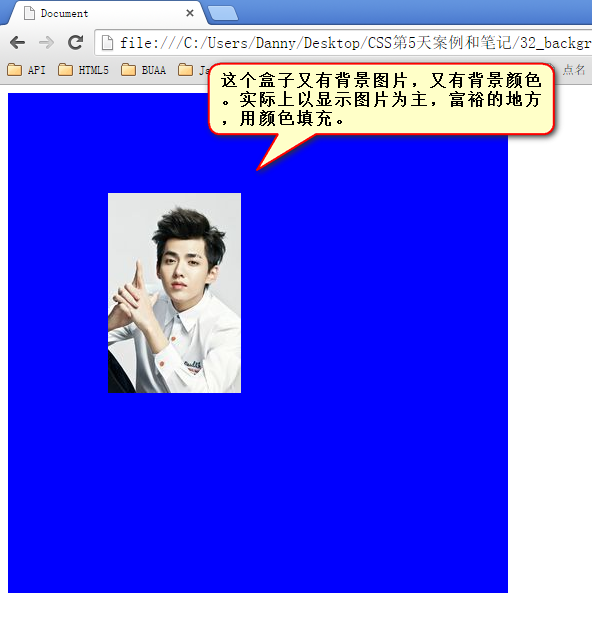
等价于：

|  |
| --- |
| 1. background-color:red; 2. background-image:url(1.jpg); 3. background-repeat:no-repeat; 4. background-position:100px 100px; 5. background-attachment:fixed; |

可以任意省略部分：

|  |
| --- |
| 1. background: red; |

|  |
| --- |
| 1. background: blue url(images/wuyifan.jpg) no-repeat 100px 100px; |



精灵的使用：

|  |
| --- |
| 1. background: url(images/taobao.png) no-repeat 0 -133px; |

预告一下，CSS3课程中，将学习更多background属性：

background-origin、background-clip、background-size（在CSS2.1背景图片是不能调整尺寸，IE9开始兼容）、

多背景。

# z-index

● z-index值表示谁压着谁。数值大的压盖住数值小的。

● 只有定位了的元素，才能有z-index值。也就是说，不管相对定位、绝对定位、固定定位，都可以使用z-index值。而浮动的东西不能用。

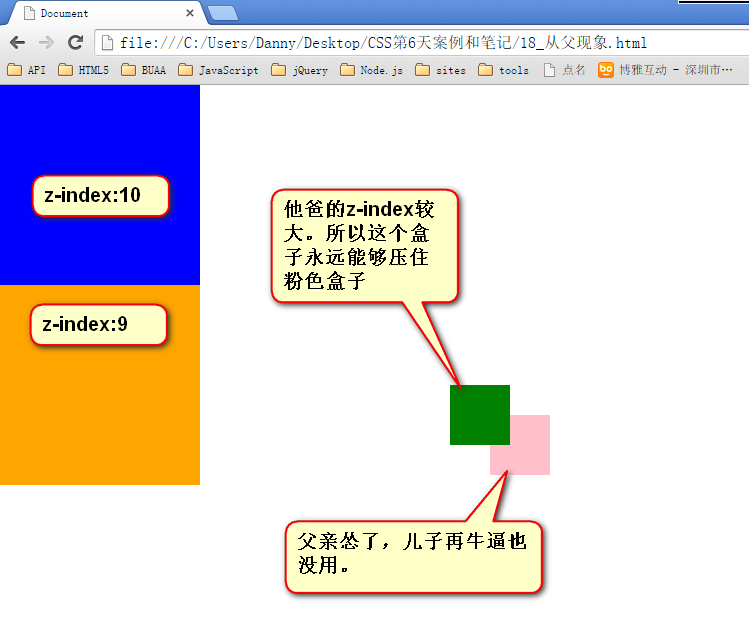
● z-index值没有单位，就是一个正整数。默认的z-index值是0。

● 如果大家都没有z-index值，或者z-index值一样，那么谁写在HTML后面，谁在上面能压住别人。定位了的元素，永远能够压住没有定位的元素。

**● 从父现象：父亲怂了，儿子再牛逼也没用。**

**没有单位：**

|  |
| --- |
| 1. z-index: 988; |

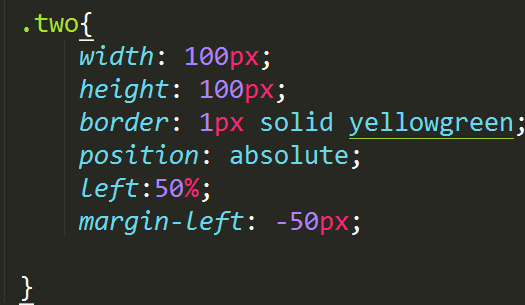


## 二十、其他新知识

### 盒子居中

Margin:0 auto; 只能让标准流下的盒子居中显示

### 定位的盒子居中显示（重要）



◆先走父盒子宽度的一半： left:50%;

◆往回走自己宽度的一半

### 标签包含规范

☞div可以包含任何标准流下的元素

☞ p标签中不能包含div 和 标题标签 及列表标签。

☞标题标签可以包含其他标签。

☞行内元素最好不要包含其他标签。

### 规避脱标流

☞网页布局过程中，能用表流布局就用标准流布局。

☞标准流不能解决用浮动

☞浮动不能解决用定位

■使用margin-left| margin-right 取值为auto可以将盒子自动冲到另一边。

### 图片与文字垂直对齐

每一种inlne-block元素 都有一个默认的vertical-align:baseline

☞vertical-align:middle 垂直对齐。 vertical-align与inline-block更搭配。

### Css可见性

☞overflow: hidden 将超出部分进行隐藏

☞display: none 直接将元素隐藏

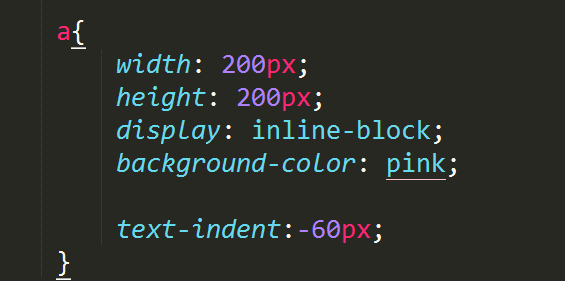
display：block 将元素显示(与js配合更搭)

☞visibility:hidden 将元素隐藏

display: none; 将元素隐藏后不占位置

visibility: hidden; 将元素隐藏后占原来的位置

### 内容移除文字（logo优化）



或者



## 鼠标样式

Cursor: pointer 鼠标变成小手

Cursor: default; 小白

Cursor : move; 移动

Cursor : text ; 文本输入

网页布局：

给一个盒子 ： 宽度高度 背景边框 位置

## 标签嵌套

1. 块级元素 --- 任何元素可以的。

2. 行内元素 -- 行内元素 只 嵌套元素 b u span i - s

3. P 不能放 div 。

4. a 无所不能 。 a里面不能放 a input 等