--

[WWW 1](#_Toc242)

[关于布局 2](#_Toc6448)

[BFC 2](#_Toc18597)

[基本概念 2](#_Toc5864)

[BFC的一个简单应用-结合float说明 2](#_Toc1706)

[+margin-top问题 3](#_Toc5746)

["无宽度"、"无图片"和"无浮动" 3](#_Toc15846)

[两栏布局常用方式：左浮动+右浮动 4](#_Toc10072)

[左浮动，右absolute padding-left 4](#_Toc24327)

[table-cell自适应布局 4](#_Toc24984)

[浮动float 4](#_Toc3065)

[基础 4](#_Toc10699)

[基础特性 4](#_Toc8161)

[float的inline-block特性 5](#_Toc7084)

[float的inline-block特性使用案例 5](#_Toc26643)

[关于float行为分析 5](#_Toc30139)

[float其他现象： 6](#_Toc7583)

[清除浮动 6](#_Toc30475)

[浮动副作用 6](#_Toc27757)

[说明: 7](#_Toc26756)

[[overflow:auto] 7](#_Toc31738)

[[方法三] 7](#_Toc5399)

[inline-block 8](#_Toc20274)

[关于inline-block 8](#_Toc20667)

[www 8](#_Toc15364)

[inline-block 前世 8](#_Toc27693)

[inline-block 今生 8](#_Toc155)

[float与 inline-block 9](#_Toc9154)

[www 9](#_Toc27496)

[float的使用局限性 10](#_Toc18903)

[inline-block的使用 10](#_Toc29784)

[空格 10](#_Toc12697)

[注意 10](#_Toc22771)

[inline-block/text-align:justify下列表的两端对齐布局 10](#_Toc17730)

[简介 10](#_Toc6707)

[1、外层元素定宽 10](#_Toc19191)

[2、margin扩大外层元素宽度 11](#_Toc29267)

[3、nowrap单行列表元素两端对齐，且不换行 11](#_Toc13810)

[4、text-align:justify实现两端对齐 11](#_Toc18928)

[demo 12](#_Toc3482)

[inline-block瀑布流布局 13](#_Toc12063)

[line-height 13](#_Toc692)

[line-height简介 13](#_Toc9847)

[定义 13](#_Toc19163)

[示意图 13](#_Toc11544)

[基础概念 13](#_Toc23764)

[行高的计算与继承 14](#_Toc15413)

[+单位 计算 继承 14](#_Toc15493)

[x浏览器的差别与错误 15](#_Toc25007)

[单行文字在垂直方向居中 15](#_Toc15249)

[多行文本居中 15](#_Toc24255)

[单行文本两端对齐 15](#_Toc31719)

[基线对齐 16](#_Toc1223)

[line-height line-block font-size height之间的关系 16](#_Toc11214)

[line-height、line-block 16](#_Toc2488)

[line-height、font-size 16](#_Toc28456)

[使用line-height代替height避免haslayout 16](#_Toc6331)

[vertical-align和line-height 16](#_Toc18442)

[说明 16](#_Toc18678)

[未知空格 17](#_Toc8411)

[原因 17](#_Toc13219)

[处理 17](#_Toc27794)

[应用：图片垂直居中（单行文本垂直居中） 18](#_Toc27616)

[特别现象 18](#_Toc13779)

[inline-block和baseline 18](#_Toc28115)

[vertical-align 19](#_Toc31733)

[问题来了 19](#_Toc2074)

[边界和baseline 20](#_Toc30009)

[inline-block元素 20](#_Toc6565)

[line-box 20](#_Toc18361)

[Vertical-align属性的值 20](#_Toc3941)

[Vertical这个属性到底是怎么起作用 20](#_Toc5851)

[下空隙 21](#_Toc10361)

[中间空隙 22](#_Toc29004)

[布局display 22](#_Toc31936)

[关于display 22](#_Toc19399)

[www 22](#_Toc24825)

[table 22](#_Toc22333)

[table布局与css布局 22](#_Toc23763)

[table-layout:fixed; 22](#_Toc13593)

[<col> 标签 23](#_Toc19106)

[基于display:table的CSS布局 23](#_Toc25260)

[概念 23](#_Toc13710)

[demo 23](#_Toc15633)

[相关属性 23](#_Toc29312)

[匿名表格元素 23](#_Toc18688)

[栅格 24](#_Toc4366)

[table-cell 24](#_Toc1641)

[display:table-cell与大小不固定元素的垂直居中 24](#_Toc27960)

[display:table-cell与两栏自适应布局 24](#_Toc5592)

[display:table-cell下的等高布局 25](#_Toc12224)

[display:table-cell下的列表布局 25](#_Toc26085)

[display:table-cell自适应布局下连续单词字符换行 25](#_Toc14924)

[关于文本换行的控制 26](#_Toc4419)

[css选择器高效使用 26](#_Toc9672)

[float + 负margin 实现自适应布局 27](#_Toc6699)

[单列定宽单列自适应布局 27](#_Toc27803)

[左右定宽中间自适应布局 27](#_Toc22067)

[两列等高自适应布局 27](#_Toc26532)

WWW

|  |  |
| --- | --- |
|  | css解决方案  <http://www.w3cplus.com/solution/index/index.html> |
| css3 layout 布局 css布局 border解决方案 border解决方案 清除浮动 清除浮动 ellipsis 省略号解决 省略号解决 图文混排解决方案 图文混排解决方案 项目列表解决方案 项目列表解决方案 新闻列表解决方案 新闻列表解决方案 css tab 标题解决方案 css tab标题解决方案 radio & checkbox 解决方案 radio/checkbox 解决方案 submit 解决方案 submit 解决方案 placeholder解决方案 placeholder解决方案 form表单解决方案 form表单解决方案 css代码copy html代码copy ie bug解决方案搜集 ie bug解决方案搜集 前端工具 前端工具 css资源宝库 css资源宝库 点滴积累 点滴积累 css3属性详解 css3属性详解教程 css3 按钮 css3按钮 css3 文字效果 css3文字效果 css3 content 生成内容 css3 生成内容 css3 box-shadow 盒子阴影 css3盒子阴影 css3 accordion tbas 特效 css3 accordion css3 accordion tbas 特效 css3 tabs |
|  | [CSS布局中一个简单的应用BFC的例子](http://www.cnblogs.com/leejersey/p/4412785.html)  <http://www.cnblogs.com/leejersey/p/4412785.html>  CSS深入理解流体特性和BFC特性下多栏自适应布局  <http://www.zhangxinxu.com/wordpress/2015/02/css-deep-understand-flow-bfc-column-two-auto-layout/>  [探究：绝对定位没有设置 top, right, bottom, left 的世界是怎样的？](https://segmentfault.com/a/1190000003109367)  <https://segmentfault.com/a/1190000003109367>  CSS基本样式库 – [zxx.lib.css](http://www.zhangxinxu.com/study/css/zxx.lib.css)  <http://www.zhangxinxu.com/study/css/zxx.lib.css>  CSS: 深入理解BFC和Margin Collapse (margin叠加或者合并外边距)  <http://justcode.ikeepstudying.com/2016/07/css-%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3bfc%E5%92%8Cmargin-collapse-margin%E5%8F%A0%E5%8A%A0%E6%88%96%E8%80%85%E5%90%88%E5%B9%B6%E5%A4%96%E8%BE%B9%E8%B7%9D/>  在线教程  <http://www.ycku.com/course/>  说说CSS学习中的瓶颈  http://www.zhangxinxu.com/wordpress/2012/07/bottleneck-css-study/ |
|  | 设计网站： 设计师网址导航：http://hao.uisdc.com/ 好123设计：http://www.hao123.com/sheji 花瓣网：http://huaban.com/ 站酷：http://www.zcool.com.cn/ 千图网：http://www.58pic.com/ 数字驿站：http://www.k1982.com/  设计达人：http://www.shejidaren.com/ 学UI网：http://www.xueui.cn/ PS资源：http://psefan.com/ps UI中国：http://www.ui.cn/ |
| 学习网： 慕课网：http://www.imooc.com/course/list 极课学院：http://www.jikexueyuan.com/ AngularJS在线课程：http://each.sinaapp.com/angular/ 前端网：http://www.w3cfuns.com/ 博客网：http://www.cnblogs.com/ React Native中文网：http://reactnative.cn/ Highcharts中文网：http://www.hcharts.cn/index.php less首页：http://www.1024i.com/demo/less/ 汇智网：http://www.hubwiz.com/ 51CTO学院：http://edu.51cto.com/ avalon中文官网：http://avalonjs.github.io/#home.html vuejs网：http://cn.vuejs.org/ |
| 资料网： 阿里巴巴矢量图标：http://www.iconfont.cn/ 图标：https://icomoon.io/app/#/select 淘宝前端团队：<http://taobaofed.org/> 腾讯ISUX：<https://isux.tencent.com/> 开源中国社区：http://www.oschina.net/ html我帮你：http://www.html580.com/ jQuery之家：http://www.htmleaf.com/ 树型图：http://showcase.ngnice.com/#/tree/treeData |
|  | 绝对定位与text-align  vertical-align和line-height  浮动和绝对定位 |
|  | [position属性absolute与relative 详解](http://www.cnblogs.com/jenry/archive/2007/07/15/818660.html)  <http://www.cnblogs.com/jenry/archive/2007/07/15/818660.html>  [CSS布局](https://segmentfault.com/bookmark/1230000005123697)  https://segmentfault.com/bookmark/1230000005123697 |
|  | avalonJS是司徒正美开发和维护的前端mvvm框架，可以轻松实现数据的隔离和双向绑定，相比angularJS等前端框架它有如下优势：  1.压缩后仅有60多kb，而angular的min版是2MB左右（无视其gzip版）；  2.兼容IE6+，符合天朝市场需求；  3.效率更高，跑起来比angular和knockout都要更快，在移动端上该优势会更大（avalon有移动端专版的avalon.modern.js）。关于其性能更详细的介绍可以看 [这里](http://www.html-js.com/article/Avalon-tutorial-22-the-performance-of-Avalon-revealed" \t "http://www.360doc.com/content/14/1031/20/_blank) ；  4.涵盖了angular的大部分功能，且实现方式更为便捷、上手更容易;  5.有配套的UI库（当然这个按需选择即可），由司徒正美及其"去哪儿"团队维护，有相关的中文文档（下方会提到），除了在github提交issue，你也可以加入正美的Q群79641290 来交流问题或提交bug。  http://avalonjs.coding.me/who.html |

关于布局

|  |  |
| --- | --- |
|  | <http://www.cnblogs.com/dojo-lzz/p/3999013.html>  《CSS设计指南》、《CSS权威指南》  <http://www.cnblogs.com/winter-cn/archive/2012/11/16/2772562.html>  <http://f2e-js.com/?p=2599>  <http://www.cnblogs.com/dolphinX/p/3508869.html>  <http://wenku.baidu.com/link?url=yRqbHnEVEL58mfPg1KDneWqX5AjcL34U70ANznTaWU6DUcTx6yaEcKBbDjPxyP3GVoNN7-GdTSPbEmty6RmCTJ3qY6FzPqSB7TvwbmFayYO>  <http://reference.sitepoint.com/css/blockformatting>  YUI 团队 CSS 101: Block Formatting Contexts <http://www.yuiblog.com/blog/2010/05/19/css-101-block-formatting-contexts/>  关于布局的一篇小总结【BFC IFC float】 <http://www.cnblogs.com/winter-cn/archive/2013/05/11/3072929.html>  CSS布局的未来  <http://www.w3cplus.com/css3/future-css-layouts.html> |

BFC

|  |  |  |
| --- | --- | --- |
| 基本概念 | 什么是BFC  BFC(Block Formatting Context)，简单讲，它是提供了一个独立布局的环境，每个BFC都遵守同一套布局规则。例如，在同一个BFC内，盒子会一个挨着一个的排，相邻盒子的间距是由margin决定且垂直方向的margin会重叠。而float和clear float也只对同一个BFC内的元素有效。  Formatting Context：指页面中的一个渲染区域，并且拥有一套渲染规则，他决定了其子元素如何定位，以及与其他元素的相互关系和作用。  BFC：块级格式化上下文，它是指一个独立的块级渲染区域，只有Block-level BOX参与，该区域拥有一套渲染规则来约束块级盒子的布局，且与区域外部无关。  什么情况产生BFC  W3C标准中这样描述：Floats, absolutely positioned elements, block containers (such as inline-blocks, table-cells, and table-captions) that are not block boxes, and block boxes with ‘overflow’ other than ‘visible’ (except when that value has been propagated to the viewport) establish new block formatting contexts for their contents.  非块级盒子的浮动元素、绝对定位元素及块级容器(比如inline-blocks，table-cells和table-captions)，以及overflow属性是visible之外任意值的块级盒子，都会创建了一个BFC。  既然上文提到BFC是一块渲染区域，那这块渲染区域到底在哪，它又是有多大，这些由生成BFC的元素决定，CSS2.1中规定满足下列CSS声明之一的元素便会生成BFC。  float：left|right  position：absolute|fixed  display：table-cell|table-caption|inline-block  overflow：hidden|scroll|auto  看到有道友文章中把display：table也认为可以生成BFC，其实这里的主要原因在于Table会默认生成一个匿名的table-cell，正是这个匿名的table-ccell生成了BFC。  BFC的约束规则  浏览器对于BFC这块区域的约束规则如下：  生成BFC元素的子元素会一个接一个的放置。垂直方向上他们的起点是一个包含块的顶部，两个相邻子元素之间的垂直距离取决于元素的margin特性。在BFC中相邻的块级元素外边距会折叠。  生成BFC元素的子元素中，每一个子元素做外边距与包含块的左边界相接触，（对于从右到左的格式化，右外边距接触右边界），即使浮动元素也是如此（尽管子元素的内容区域会由于浮动而压缩），除非这个子元素也创建了一个新的BFC（如它自身也是一个浮动元素）。  有道友对它做了[分解](http://f2e-js.com/?p=2599" \t "http://www.cnblogs.com/dojo-lzz/p/_blank)，我们直接拿来：  > 内部的Box会在垂直方向上一个接一个的放置  > 垂直方向上的距离由margin决定。（完整的说法是：属于同一个BFC的两个相邻Box的margin会发生重叠，与方向无关。）  > 每个元素的左外边距与包含块的左边界相接触（从左向右），即使浮动元素也是如此。（这说明BFC中子元素不会超出他的包含块，而position为absolute的元素可以超出他的包含块边界）  > BFC的区域不会与float的元素区域重叠  > 计算BFC的高度时，浮动子元素也参与计算  > BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面元素，反之亦然  看到以上的几条约束，让我想起学习css时的几条规则  > Block元素会扩展到与父元素同宽，所以block元素会垂直排列  > 垂直方向上的两个相邻DIV的margin会重叠，而水平方向不会(此规则并不完全正确)  > 浮动元素会尽量接近往左上方（或右上方）  > 为父元素设置overflow：hidden或浮动父元素，则会包含浮动元素  ...... | |
|  | 初探BFC  我们先看看CSS布局的基本单位：  block-level box: display 属性为 block, list-item, table 的元素，会生成 block-level box，并且参与 block formatting context  inline-level box: display 属性为 inline, inline-block, inline-table 的元素，会生成 inline-level box，并且参与 inline formatting context  那这个formatting context又是什么玩意儿？  它是W3C CSS2.1规范中的一个概念。它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。也就是说我们平时在布局的时候，它默默地提供了一个环境，使得HTML元素在这个环境中按照一定规则进行布局。  最常见的formatting context有Block Formatting Context(BFC)和Inline Formatting Context(IFC)，CSS3中还增加了GridLayout Formatting Context(GFC)和Flex Formatting Context(FFC)，这里就不深入研究了。  BFC定义：它是一个独立的渲染区域，只有block-level box参与，其中的元素按照规定的渲染规则进行布局，并且这个区域与外部毫不相干。 | |
| 应用 | 1.防止垂直margin重叠  有点布局经验的朋友都知道margin collapse，也就是相邻的垂直元素同时设置了margin后，实际margin值会塌陷到其中较大的那个值。其根本原理就是它们处于同一个BFC，符合“属于同一个BFC的两个相邻元素的margin会发生重叠”的规则。  我们可以在其中一个元素外面包裹一层容器，并触发该容器生成一个BFC。那么两个元素便属于不同的BFC，就不会发生margin重叠了：  2.防止浮动子元素高度塌陷  <div class="parent">  <div class="child">float: left</div>  <div class="child">float: left</div>  </div>  如果我们将.parent元素的overflow: hidden去掉，那么.parent元素就获取不到浮动元素的高度了。但是加上overflow属性后触发了BFC，计算BFC的高度时，浮动元素也参与了计算。  3.防止文字（或其他元素）环绕  <div class="parent">  <div class="child">float: left</div>  <div class="text">我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕我只是文字但我不会环绕</div>  </div>  正常情况下，如果一个块级元素设置成了float，那么他的兄弟元素会环绕其布局。这里我们给.text加上overflow，文字所在的区域就产生了BFC，元素的左边总是触碰到容器的左边，即使存在浮动也是如此。  这里举例了几个比较经典的应用，我们在以后的布局中也可以慢慢探索其中的奥秘之处，做一个能写一手好样式还能懂原理的前端er。  原文链接：<https://segmentfault.com/a/1190000011211625> | |
|  |  | |
| BFC的一个简单应用-结合float说明 | [CSS布局中一个简单的应用BFC的例子](http://www.aliued.cn/2012/12/31/css%e5%b8%83%e5%b1%80%e4%b8%ad%e4%b8%80%e4%b8%aa%e7%ae%80%e5%8d%95%e7%9a%84%e5%ba%94%e7%94%a8bfc%e7%9a%84%e4%be%8b%e5%ad%90.html" \o " CSS布局中一个简单的应用BFC的例子)  <http://www.aliued.cn/2012/12/31/css%E5%B8%83%E5%B1%80%E4%B8%AD%E4%B8%80%E4%B8%AA%E7%AE%80%E5%8D%95%E7%9A%84%E5%BA%94%E7%94%A8bfc%E7%9A%84%E4%BE%8B%E5%AD%90.html>  CSS深入理解流体特性和BFC特性下多栏自适应布局  <http://www.zhangxinxu.com/wordpress/2015/02/css-deep-understand-flow-bfc-column-two-auto-layout/> | |
| <**style**>  **/\*1：此处的border与outline效果不同 > margin-top与border结合有诡异效果；相邻元素的 margin 重叠问题， 你设置了border,为了显示border ，就不会重叠了 \*/** .item {  width: 300px;  **/\*1\*/** position: absolute;  **/\*overflow: hidden;\*/  /\*border: 1px solid #b2d1a3;\*/  /\*outline: 1px solid red;\*/** background-color: #e5ebe4;  }  .pic {  width: 80px;  height: 80px;  margin: 10px;  color: #fff;  background-color: #acdae5;  }  .cont {  margin: 10px;  color: #37a;  } </**style**> <**div** class='item'>  <**div** class='pic'>picture</**div**>  <**p** class='cont'>  爱饭否，爱豆瓣，也爱鸡脱壳。 爱爬山，爱拍美景。 爱腐败，更爱远征的自虐。 爱下雪天，爱感动，爱平底鞋。 我没有什么特别，我很特别。 我和别人不一样，我和你一样。 我是前端。  </**p**> </**div**> | > 这段代码是结构简单的三个元素的叠放，效果如下：  > 1：此处的border与outline效果不同： margin-top与border结合有诡异效果；相邻元素的 margin 重叠问题， 设置了border,为了显示border ，就不会重叠了 |
|  | 这时，如果要将文字部分放到图片的右侧，很多人都会想到给.pic使用float:图1  右侧内容并没有如我们预料一样规整的排在右侧，而是将左侧图片包围起来。接下来，我们为右侧内容部分设置overflow:hidden属性来使它形成一个新的BFC：图2  这确实是我们想要的  可是，在IE6下看到的却是下面这样：图3  不仅内容区没有排在右侧，而且遭遇了双边距bug。双边距bug非本文重点讨论，直接为.pic增加display:inline来解决。  我们关心的是为什么IE6下右侧内容元素还是不能决定自己的布局呢？  这里涉及到了另一个概念HasLayout。其实，在完美世界的字典里，是没有HasLayout这个概念的。它是IE浏览器引擎内部特有的属性，它可以影响到元素的定位和元素之间的相互作用。当一个元素的HasLayout属性为true时，这个元素才可以决定自己和其子孙元素的布局。为数不多的元素默认这个属性值为true，包括：  body and html  table, tr, th, td  img  hr  input, button, file, select, textarea, fieldset  marquee  frameset, frame, iframe  objects, applets, embed  所以，当发现有些元素的布局在IE下有异常时，可以有充分的理由来怀疑可能是hasLayout属性为false。而且这个属性值不能直接设置。一个元素要么默认拥有，要么通过设置特定的CSS属性来获取。直接的使元素hasLayout属性值为true的方法是声明下面的CSS属性之一：  width: 除auto之外的值  height: 除auto之外的值  float: left|right  position: absolute  display: inline-block  writing-mode: tb-rl(IE)  zoom: 除normal之外的值  IE7中增加了一些同样效果的属性：  min-height: 任意值  max-height: 除 "none" 之外的任意值  min-width: 任意值  max-width: 除 "none" 之外的任意值  overflow: hidden|scroll|auto  overflow-x: hidden|scroll|auto  overflow-y: hidden|scroll|auto  position: fixed  最常用的是zoom:1，因为这个设置对元素外观不会造成任何影响。但是这个属性是IE特有的CSS属性，不会通过CSS检查器([W3C提供的CSS校验器](http://jigsaw.w3.org/css-validator/validator" \t "http://www.cnblogs.com/leejersey/p/_blank))（当然，让不让通过校验实际取决于各种校验器的规则）。所以，有推荐对于IE7，最好是设置min-height:0。因为0是min-height的初始值，这样不会对元素外观造成影响。对于IE6及更早的版本，推荐方法是设置height:1%。这个高度会使得容器盒子的大小刚好包含内容区而忽略掉真正的属性值。但这个设置的缺陷是会影响到其他浏览器的解析，因此需要使用hack屏蔽掉对其他浏览器的影响。  所以，最终的代码可能是： | |
| <**style**>  .box {  width: 300px;  border: 1px solid #b2d1a3;**/\*\*/** background-color: #e5ebe4;  }  .pic {  width: 80px;  height: 80px;  margin: 10px;  color: #fff;  float: left;**/\*\*/**  display: inline;  }  .cont {  margin: 10px;  color: #37a;  overflow: hidden;**/\*\*/** \_height: 1%;**/\*\*/** } </**style**> <**div** class='box'>  <**div** class='pic'>picture</**div**>  <**div** class='cont'>  爱饭否，爱豆瓣，也爱鸡脱壳。 爱爬山，爱拍美景。 爱腐败，更爱远征的自虐。 爱下雪天，爱感动，爱平底鞋。 我没有什么特别，我很特别。 我和别人不一样，我和你一样。 我是前端。  </**div**> </**div**> | 左侧float    左侧float 右侧overflow hidden    ie特别情况 |
| +margin-top问题 | 父元素与子元素之间的margin-top问题(css hack)  当你想让两个元素的content在垂直方向(vertically)分隔时，既可以选择padding-top/bottom，也可以选择margin-top/bottom，再此[Ruthless](http://www.cnblogs.com/linjiqin/" \t "http://www.cnblogs.com/wzhiq896/p/_blank)建议你尽量使用padding-top/bottom来达到你的目的，这是因为css中存在Collapsing margins(折叠的margins)的现象。  Collapsing margins: margins折叠现象只存在于临近或有从属关系的元素，垂直方向的margin中。  给子元素盒子一个垂直外边距margin-top,父元素盒子也会往下走margin-top的值，遇到此问题的朋友可以参考下本文或许会有意想不到的收获  hack：  父元素的盒子包含一个子元素盒子，给子元素盒子一个垂直外边距margin-top,父元素盒子也会往下走margin-top的值，而子元素和父元素的边距则没有发生变化。  <**style**>  .box1 {  height: 200px;  width: 200px;  background: gray;  }  .box2 {  height: 100px;  width: 100px;  background: gold;  margin-top: 50px;  }    </**style**> <**div** class="box1">   <**div** class="box2"></**div**> </**div**>  解决方法：  > 修改父元素的高度，增加padding-top样式模拟（padding-top:1px;常用）  > 为父元素添加overflow:hidden;样式即可（完美）  > 为父元素或者子元素声明浮动（float:left;可用）  > 为父元素添加border（border:1px solid transparent可用）  > 为父元素或者子元素声明绝对定位 | margin：层的边框以外留的空白  background-color：背景颜色  background-image：背景图片  padding：层的边框到层的内容之间的空白  border：边框  content：内容  > <http://www.cnblogs.com/wzhiq896/p/6020329.html> |

"无宽度"、"无图片"和"无浮动"

|  |  |  |  |
| --- | --- | --- | --- |
|  | <http://www.zhangxinxu.com/wordpress/2010/10/%E9%A1%B5%E9%9D%A2%E9%87%8D%E6%9E%84%E2%80%9C%E9%91%AB%E4%B8%89%E6%97%A0%E5%87%86%E5%88%99%E2%80%9D-%E4%B9%8B%E2%80%9C%E6%97%A0%E5%AE%BD%E5%BA%A6%E2%80%9D%E5%87%86%E5%88%99/>  页面重构"鑫三无准则" 之"无宽度"准则 | | |
|  | 一、关于"鑫三无准则"  "鑫三无准则"这个概念早在去年我写"[关于Google圆角高光高宽自适应按钮及其拓展](http://www.zhangxinxu.com/wordpress/?p=292)"一文时就已经提到了。这是自己在页面重构的经验中总结出来的一套约束自己CSS的准则，即"无宽度"、"无图片"和"无浮动"，目的是使CSS布局模块化以及增强可扩展性。  此准则是针对我个人的，可能没有什么适用性，也可能会对您的学习有所启发，所以这里还是简单分享下。共三个准则，每个准则内容都较多，本文就只简单讲讲"无宽度准则"。  二、为什么要"无宽度"  先说点题外话。我个人非常不喜欢"切图"一词，每次项目经理叫我："XX，麻烦把这张图切一下"的时候，心里总是有点不畅快。"切图"一词有将设计图切割拼接之意，我们或许都经历过这类"切图"的阶段，这是个缺少创造，缺少艺术，缺少技术含量的阶段，这是个犹如搭积木般只需要幼儿园智商就能完工的阶段。  一提到"切图"一词，我脑中闪现的就是：分割图片，量体裁衣，计算宽度，页面元素无缝拼接，然后形成页面。我刚接触CSS那会儿就是这么做的，现在回首看看，当时的那种"切图"的做法真是差劲得连拜月教主都不如。  具体描述当时的做法，就是精确测量、计算每个元素的宽度，计算的时候还考虑到margin，padding以及border，总是精确到像素级，然后使用浮动进行无缝拼接，哇咔咔，当时的我还是很沾沾自喜的，因为页面上的宽度啊什么的都是精确到每一像素，每块元素都进行准确的计算，是多么的认真、工作多么的努力啊！ 现在看来，当时的那些"认真"、"努力"什么的都是浮云，完全就是公公背儿媳妇过河——吃力不讨好！  当时的那种做法就称之为"切图"，是很初级和急需提高的阶段。为什么说这种"切图"式的做法很低级呢？一是开发效率低下（要计算宽度什么的）；二是结构的重用性和可扩展性就是个高邮咸鸭蛋；三是页面的日后可维护性就是坨熊猫便便（错位等bug像要债的一样三天两头串门）。  打个形象的比方，这类宽度计算元素拼接的做法就好比用坚固的砖头砌房子，一块一块的摞起来。最后的页面看似紧密坚固，但是一旦地震一来，"房子"会以迅雷不及掩耳的速度垮掉，而那些横版的木制房却总能在强震中幸存下来。你有没有注意到，每当台风来的时候，最先倒掉的都是平时那些看似坚固的电线杆，但是却从未听说过柳树被风吹倒的。  使用宽度计算+横向拼接的做法不仅浪费精力、开发成本，最终实现的页面其实也是比较脆弱的。  宽度限定，左右浮动，这也是目前主流布局方式。但是主流的方法并不一定是最好的方法，就此例子而言，没有左右padding/margin/border等属性，所以右侧栏的宽度(484px)确定貌似还是比较简单的（事实上还是要去量一下，这是可避免的劳力成本）；但是，其扩展性和重用性则完全不及格了，首先，这里的列表不可能放在500像素宽的div中（没有了重用性），日后要是网站改版，总体宽度增加，那么显然，这里的宽度又要重新计算，这就缺少了扩展性；同时，这种浮动布局会带来一些副作用，首先是浮动本身的代码成本，其次是清除浮动所需要的成本（见下图）。 //zxx:这里的width:100%让IE6/7下li元素haslayout，可修复浮动塌陷的问题，但是个人是不推荐这种做法，要是日后此标签需要增加左右的margin/padding值，那就有活干了。  所以，如果以CSS模块化以及重用性为基本要求的话，这里的布局显然要"无宽度"，最大限度的利用标签本身的特性，这样，CSS代码成本又小，又利于日后的扩展与维护，何乐而不为呢？ 我自己归结的这个"无宽度"准则并不是自己随便YY臆想出来的，是时刻以"模块化和重用性"作为页面布局基本要求下的必然产物。固定的宽度值限制了页面内容的重用性和扩展性，所以，要想页面布局具有较高的重用性和扩展性，我们需要遵循"无宽度"准则。  我们究竟是在"页面切图"还是在"页面重构"，或许就体现在一些小的细节与实现方式上。 | | |
| 两栏布局常用方式：左浮动+右浮动 | <**style**>  .out {  height: 500px;  width: 388px;**/\*388 288 588\*/** }  .leftDiv {  float: left;  width: 88px;  height: 100px;  }  .rightDiv {  float: right;  width: 300px;  } </**style**> <**div** class="out">  <**div** class="leftDiv">  <**img** src="images/1.jpg" alt="1" width="88"/>  </**div**>  <**div** class="rightDiv">  定宽的浮动布局。左边 左浮动；右侧 右浮动  <**br**/>  <**img** src="images/2.jpg"/>  </**div**> </**div**> | | |
| 通过左右定宽浮动实现常见的两栏布局，会受限于外层元素的宽度；  正常：图1  外层div的宽度大于内层左右浮动元素的总宽度，过度张开：图2  外层div的宽度小于内层两左右浮动元素的总宽度，出现叠层：图3 | | |
| 左浮动，右absolute padding-left | 未设定left/top值的absolute元素和float元素就是两个混蛋近亲，同样的都是"包裹与破坏"，但是，虽然很多时候，这种破坏会给我们带来些麻烦（例如清除浮动造成的高度塌陷的问题），但是，万物皆有两面性，我们有时候可以利用这种破坏的特性来帮助我们进行更加有韧性的布局。  由于让左边的头像浮动(float:left)在IE6下会有神奇的3像素bug，所以，我个人偏好于使用position:absolute属性，只要这一个属性就可以了，然后，右边的内容就可以当这个头像不存在进行布局； | | |
| <**style**>  .out {  height: 500px;  position: relative;/\*控制内层元素不超出边界\*/  }  .leftDiv {  float: left;  width: 100px;  }  .rightDiv {  position: absolute;/\*absolute貌似无用，真正能起到作用的是padding-left\*/  padding-left: 100px;  }  </**style**> <**div** class="out">  <**div** class="leftDiv">  <**img** src="images/1.jpg" alt="" height="80"/>  </**div**>  <**div** class="rightDiv">  定宽的浮动布局。左边头像定宽左浮动 ；右侧内容 不定宽  定宽的浮动布局。左边头像定宽左浮动 ；右侧内容 不定宽  <**br**/>  <**img** src="images/2.jpg"/>  </**div**> </**div**> |  | |
| table-cell自适应布局 | float外加table-cell方法（table-cell的宽度上千上万却依然布局良好，很神奇），可能是因为国外基本上不鸟IE6浏览器的缘故，所以会果敢的使用display:table-cell属性；  .out(浏览器的宽度)与.leftDiv宽度变换不会影响整体布局的变换; | | |
| <**style**>  .out {  height: 500px;  }  .leftDiv {  float: left;  width: 100px;  }  .rightDiv {  display: table-cell;  vertical-align: top; **/\*当设置height值，且其值大于内部元素的总高度，将起作用\*/** width: 800px; **/\*具体值是动态变化的；由内元素、自定义与外元素共同决定；此处定义的800相当于最宽值，\*/** } </**style**> <**div** class="out">  <**div** class="leftDiv">  <**img** height="80" src="images/2.jpg"/>  </**div**>  <**div** class="rightDiv">  定宽的浮动布局。左边头像左浮动，右侧内容右浮动  <**br**/>  <**img** src="images/2.jpg"/>  </**div**> </**div**> | |  |
| <**style**>  .fix {  \*zoom: 1;  }  .fix:after {  display: block;  content: **"clear"**;  height: 0;  clear: both;  visibility: hidden;  }  .l {  float: left;  }  .cell {  display: table-cell;  \*display: inline-block;  width: 2000px;  \*width: auto;  } </**style**> <**div** class="fix">  <**div** class="l">  <**img** src="images/1.jpg" alt="1"/>  </**div**>  <**div** class="cell">  <**img** src="images/2.jpg" alt="2"/>  <**br**/>  无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、  </**div**> </**div**> | | 兼容写法 |

浮动float

基础

|  |  |  |  |
| --- | --- | --- | --- |
| 基础特性 | **浮动是个行级的行为，当遇到浮动元素的时候，会首先"假装"它是个行内元素进行排版，排好后就往浮动的方向挤到挤不过去为止（遇到边界或者其它浮动元素）**  ===  元素被标注float后，将具备inline-block（block）属性:  > span对height、width起作用  > div元素未换行  ===  > 关于float：float的直接用意：让文字环绕图片 | | |
| float的inline-block特性 | <**style**>  .testA {  float: left;  width: 300px;  height: 100px;  border: 1px solid #0000ff;  } </**style**> 文本1<**span** class="testA">文本2</**span**>文本3 | | |
| <**style**>  .testB{  float: left;  } </**style**> <**div**>111</**div**> <**div** class="testB">222</**div**> <**div**>333</**div**> | | |
| float的inline-block特性使用案例 | 宽度自适应按钮组：实现宽度自适应的关键就是要让按钮的大小自适应于文字的个数，这就需要按钮要自动包裹在文字的外面。我们用什么方法实现呢？  一就是display:inline-block；  二就是float。    <**style**>  .btn1 {  outline: 1px solid red;  display: inline-block;  padding-left: 3px;  color: #000000;  font-size: 12px;  text-decoration: none;  }   .btn1 **cite** {  display: block;  line-height: 26px;  padding: 0 13px 0 10px;  } </**style**> <**a** href="javascript:void(0);" class="btn1"><**cite**>inline-block方法</**cite**></**a**> <**a** href="javascript:void(0);" class="btn1"><**cite**>inline-block方法</**cite**></**a**>    <**style**>  .btn2 {  outline: 1px solid red;  float: left;  padding-left: 3px;  color: #000000;  font-size: 12px;  text-decoration: none;  }   .btn2 **cite** {  display: block;  line-height: 26px;  padding: 0 13px 0 10px;  } </**style**> <**a** href="javascript:void(0);" class="btn2"><**cite**>float方法</**cite**></**a**> <**a** href="javascript:void(0);" class="btn2"><**cite**>float方法</**cite**></**a**> | | |
| 关于float行为分析 | <**div** style="">  jian dan wen ben 文本1 <**img** style="" src="images/ttt.jpg" alt=""/>文本2  <**br**/>  文本3 <**img** src="images/ttt.jpg" alt=""/>文本4 </**div**> | | |
| <**div** style="">  <**span**>文本1</**span**>  <**img** class="" src="images/ttt.jpg" alt=""/>  <**span**>文本2</**span**>  <**br**/>  <**span**>文本3</**span**>  <**img** src="images/ttt.jpg" alt=""/>  <**span**>文本4</**span**> </**div**> | |  |
| <**div** style="">  <**span**>文本1</**span**>  <**img** class="fl" src="images/2.jpg" alt=""/>  <**span**>文本2</**span**>  <**br**/>  <**span**>文本3</**span**>  <**img** src="images/ttt.jpg" alt=""/>  <**span**>文本4</**span**> </**div**> | |  |
|  | <**span**>文本1</**span**> <**img** class="fl" src="images/ttt.jpg" alt=""/> <**span**>文本2</**span**> <**span**>文本3</**span**> | |  |
|  | <**span**>文本1</**span**> <**img** class="fl" src="images/ttt.jpg" alt=""/> <**span**>文本2</**span**> <**br**/> <**span**>文本3</**span**> <**img** src="images/ttt.jpg" alt=""/> <**span**>文本4</**span**> | |  |
| <**div** style="width: 300px">  <**span**>文本1 文本1 文本1 文本1</**span**>  <**img** style="float: left;" src="images/ttt.jpg" alt=""/>  <**span**>文本2</**span**>  <**span**>无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、</**span**> </**div**> | |  |
| float其他现象： | ===  关于在line中的inline[inline、inline-block]元素使用float：  > 用意：一个line元素中的存在几个inline元素，将其中标注"float:left;"的元素中排到最前面；是横向；  > 注意：如果一行最多只能存在一个inline元素，那么float将无意义（如下）（当第一个span与img在一行中不能一起显示时，floa img的前移特性将无效）（float是相对于一行中的前元素而言的）； | | |
| <**div** style="width: 600px">  文 本 1 文 本 1 文 本 1  <**img** style="float: left;" src="images/ttt.jpg" alt=""/>  文 本 2 文 本 2 文 本 2 </**div**> | 当外层div的宽度逐渐变小时： | |
|  |  | |
|  |  | |
|  |  | |
|  | <**style**>  .one{  display: inline-block;  块元素无效  行内元素效果一样   width: 100px;  height: 100px;  background: linen;  }  .two{  float: left;   width: 300px;  height: 100px;  background: #0000ff;  } </**style**>  <**div** class="one">1</**div**> <**div** class="two">2</**div**> <**div** class="two">3</**div**> <**div** class="two">4</**div**> |  | |

清除浮动

|  |  |
| --- | --- |
| 浮动副作用 | <**style**>  .fl {  float: left;  }  .test {  width: 100px;  height: 100px;  } </**style**> <**div** style="padding: 10px;background: lightsalmon;">  <**div** class="fl test">1</**div**>  <**div** class="fl test">2</**div**>  <**span**>文字</**span**> </**div**> <**span**>文字排版异常2</**span**> <**img** src="images/1.jpg" alt=""/> |
| 1、背景不能显示 由于浮动产生，如果对父级设置了（[CSS background背景](http://www.divcss5.com/rumen/r125.shtml)）[CSS背景颜色](http://www.divcss5.com/jiqiao/j369.shtml)或[CSS背景图片](http://www.divcss5.com/jiqiao/j369.shtml)，而父级不能被撑开，所以导致[CSS背景](http://www.divcss5.com/rumen/r125.shtml)不能显示。  2、边框不能撑开 如上图中，如果父级设置了[CSS边框](http://www.divcss5.com/rumen/r120.shtml)属性（[css border](http://www.divcss5.com/rumen/r120.shtml)），由于子级里使用了float属性，产生浮动，父级不能被撑开，导致边框不能随内容而被撑开。  3、margin padding设置值不能正确显示 由于浮动导致[父级子级](http://www.divcss5.com/rumen/r239.shtml)之间设置了css padding、[css margin](http://www.divcss5.com/rumen/r128.shtml)属性的值不能正确表达。特别是上下边的padding和[margin](http://www.divcss5.com/rumen/r128.shtml)不能正确显示。 |
| 方法1 | 对父级设置适合CSS高度：使父元素的高度超过所有子元素的高度  <**style**>  .fl {  float: left;  }  .test {  width: 100px;  height: 100px;  } </**style**> <**div** style="padding: 10px;background: lightsalmon;height: 100px">  <**div** class="fl test">1</**div**>  <**div** class="fl test">2</**div**>  <**span**>文字</**span**> </**div**> <**span**>文字排版异常2</**span**> <**img** src="images/1.jpg" alt=""/> |
| 方法2 | .clear{clear:both;}  [现在可能还可以在很多老的站点上可以看到这样的代码，相当暴力有效的解决浮动的问题。但是这个用法有一个致命伤，就是每次清除浮动的时候都需要增加一个空标签来使用。]  <**style**>  .clear{  clear: both;  } </**style**> <**div**>  <**div** class="left"></**div**>  <**div** class="right"></**div**>  <**div** class="clear"></**div**>**<!--父级元素标签结束前使用--><!--会增加一个空标签-->** </**div**>  > clear语法：  clear:none left right both  > clear参数说明：  none：允许两边都可以有浮动对象  both：不予许有浮动对象  left：不予许左边有浮动对象  right：不允许右边有浮动对象 |
| 方法3 | 父元素添加overflow:hidden;或者overflow:auto;  另外在 IE6 中还需要触发 hasLayout ，例如为父元素设置容器宽高或设置 zoom:1  在添加overflow属性后，浮动元素又回到了容器层，把容器高度撑起，达到了清理浮动的效果。  那是因为overflow:hidden属性相当于是让父级紧贴内容，这样即可紧贴其对象内内容（包括使用float的div盒子），从而实现了清除浮动。Css overflow:hidden清除浮动方法推荐使用。  缺点：对margin属性会有影响，不能设负值，会被裁掉。负值绝对定位也不可以。  zoom:1;属性是IE浏览器的专有属性，Firefox等其它浏览器不支持。它可以设置或检索对象的缩放比例。除此之外，它还有其他一些小作用，比如触发ie的hasLayout属性，清除浮动、清除margin的重叠等。 |
| 方法4 | .clearfix:after{  content: "\020";  display: block;  height: 0;  clear: both;  visibility: hidden;  }  .clearfix {  \*zoom: 1; /\* 触发 hasLayout \*/ /\*IE/7/6\*/  }  [结合 :after 伪元素（注意这不是伪类，而是伪元素，代表一个元素之后最近的元素）和 IEhack ，可以完美兼容当前主流的各大浏览器，这里的 IEhack 指的是触发 hasLayout]  给浮动元素的容器添加一个clearfix的1class，然后给这个class添加一个:after伪元素实现元素末尾添加一个看不见的块元素（Block element）清理浮动]  [zoom:1;属性是IE浏览器的专有属性，Firefox等其它浏览器不支持。它可以设置或检索对象的缩放比例。除此之外，它还有其他一些小作用，比如触发ie的hasLayout属性，清除浮动、清除margin的重叠等。]  \* html .clearfix { zoom: 1; } /\* IE6 \*/  \*:first-child+html .clearfix { zoom: 1; } /\* IE7 \*/  [IE6 和 IE7 都不支持 :after 这个伪类，因此需要后面两条来触发IE6/7的haslayout，以清除浮动。幸运的是IE8支持 :after 伪类。因此只需要针对IE6/7的hack了]  .clearfix:after {  content:"\200B";  display:block;  height:0;  clear:both;  }  .clearfix {\*zoom:1;}/\*IE/7/6\*/  [解释下:content:"\200B";这个参数，Unicode字符里有一个"零宽度空格"，即 U+200B，代替原来的"."，可以缩减代码量。而且不再使用visibility:hidden。]  .clearfix:before,.clearfix:after{  content:"";  display:table;  }  .clearfix:after{clear:both;}  .clearfix{  \*zoom:1;/\*IE/7/6\*/  } |
|  |  |
| 说明:  [overflow:auto]  [方法三] | 演示1：      下面的演示中显示的浮动子元素在父容器高度不自动适应的问题 。为了解决这个问题，您可以简单地添加CSS属性overflow:auto (or overflow:hidden)的包装容器。这也许是最简单的方法来清除浮动。  [IMG_256](http://webdesignerwall.com/demo/clear-float/)  .container {  overflow: auto; } |
| 演示2：  Overflow:auto 也可以用来防止从周围的浮动元素包装的内容。比方说，你正在设计一个评论列表。您将最有可能，有一个头像，浮于左，评论在右。要防止的头像周围环绕的评论，只需添加 overflow:hidden 的评论容器。使用overflow这里的优点是，我没有给评论容器设置一个float or width。容器自动校准浮动的头像图片。  [IMG_257](http://webdesignerwall.com/demo/clear-float/demo2.html)  .image {  float: left; }  .text {  overflow: hidden; } |
| 缺点:  虽然它是一个很好的的技巧，也有一些弊端：  使用 overflow:auto，会造成一个滚动条，如果您的内容是延长了容器的边界。例如，如果你有一个长unbreaking的文本（即长的URL文本）或一个较大的图像，是更大的，则容器滚动显示。  为了避免一个滚动显示你应该使用overflow:hidden。然而，这种方法也有一个缺点。使用overflow:hidden隐藏任何超出容器的边界的内容。  Word-wrap  为了解决大文本问题，只需添加 word-wrap:break-word 到容器，这将迫使文本换行到一个新的行。  .container {  word-wrap: break-word; }  Max-width  为了防止图像扩大超出容器边界，添加的max-width:100%，它会调整图像的大小符合容器的最大宽度。  .container img {  max-width: 100%;  height: auto; } |
| 结论： | <**style**>  **/\*1\*/** .clear-fix {  \*overflow: hidden;  \*zoom: 1;  }  .clear-fix:after {  display: table;  content: **""**;  width: 0;  clear: both;  }  **/\*2\*/** .fix {  \*zoom: 1;  }  .fix:after {  display: block;  content: **"clear"**;  height: 0;  clear: both;  visibility: hidden;  } </**style**> |

inline-block

关于inline-block

|  |  |
| --- | --- |
| www | <http://forsigner.com/2013/03/28/inline-block/> |
| inline-block 前世 | 1.认知  也许有人问你为何要写「 display:inline-block; display:inline; zoom:1; 」 来兼容 IE6、7 时，你会立马答道：因为 IE6、7 不支持 display:inline-block 呗!不知道何时起，惯性思维给开发者带来了这样一个可怕的概念。万物都是辩证的，当你写下这些的时候，可曾怀疑过大众观点真的可靠吗？也许你认为这些无关 紧要，实现效果就好。但是如果不能理解每个属性或属性值的根本，你将永远无法全面的了解它，人云亦云只会让你浅尝辄止，止步不前。那么这里就涉及到所谓的 「CSS 学习瓶颈」的问题了，这个问题张鑫旭《说说CSS 学习中的瓶颈》一文有详细阐述，虽然部分观点我不是很赞同，但是中心思想还是很值得思考的。文中有几个不错的问题这里也列举出来供大家观摩：  line-height:150% 和 line-height:1.5 的区别是？  float 为何会让外部容器高度塌陷？这是 bug？（我的答案在《那些年我们一起清除过的浮动》）  vertical-align 的表现为何在IE7, IE8, IE9 下表现不尽相同？其中的渲染机制是？ 好了，回到 inline-block 的认知的问题，我的观点是：  IE 从 5.5 开始就已经支持 display:inline-block 了，只是支持的并不是那么完善。  在 msdn 微软开发者社区，找到了 IE 从5.5 开始支持 inline-block 的证据：  The inline-block value is supported starting with Internet Explorer 5.5. You can use this value to give an object a layout without specifying the object’s height or width.  这里明确指出：从 IE5.5 开始支持 inline-block。  链接：[http://msdn.microsoft.com/zh-cn/library/ie/ms530751(v=vs.85).aspx](http://msdn.microsoft.com/zh-cn/library/ie/ms530751(v=vs.85).aspx" \t "http://forsigner.com/2013/03/28/inline-block/_blank)  那么既然 IE5.5 开始就已经支持了 inline-block，为何我们还要写那么一坨 CSS 呢？同时我们知道 IE6、7 中 display:inline-block 是可以触发 hasLayout 的，触发了 hasLayout 的元素表现出来的特征就是一个独立的矩形容器，可以设置宽高而且不受外部元素的影响，类似于现代浏览器中的 Block formatting contexts （块级格式化上下文）的概念。  下面来做一个详细的测试，分别看看 IE6 中 inline 元素和 block 元素的表现：  1）inline 元素 display:inline-block   |  | | --- | | .dib-inline, .dib-block {  width:100px;  height:30px;  line-height:30px;  text-align:center; } .dib-inline {  display:inline-block; } |   测试表明：IE6 中 inline 元素只要触发了 hasLayout 其表现就类似于 inline-block，这里设置 display:inline-block; 或者 zoom:1; 等其他属性值可以触发 hasLayout ，表现出来是一样的。  2）block 元素 display:inline-block  IE6 中截图如下：   |  | | --- | | .dib-inline, .dib-block {  width:100px;  height:30px;  line-height:30px;  text-align:center; } .dib-block{  display:inline-block; } |   测试表明：IE6 中 block 元素即使触发了 hasLayout 也不能具有 inline-block 元素不换行的特性。想要 block 元素支持 inline-block 元素的特性，我们可以这样做：   |  | | --- | | .dib-block {  display:inline;  zoom:1; } |   首先让 block 元素转化为 inline 元素，强制其不换行；然后通过 zoom:1 触发 hasLayout，使其可以设置宽高。修复后的 截图如下：  3）结合现代浏览器  综上，现代浏览器都支持 display:inline-block ，IE6、7 inline 元素也可以达到同样的效果，IE6、7 block 元素需要设置 display:inline; zoom:1; 它们结合在一起便是：   |  | | --- | | display:inline-block; /\* 现代浏览器 +IE6、7 inline 元素 \* \*display:inline; /\* IE6、7 block 元素 \* \*zoom:1; |   为了不让支持 CSS2.1 inline-block 的浏览器 重置为 inline，我们针对 IE6、7 做一个 hack。由于现代浏览器也开始支持 zoom 属性，这里只是希望 IE6、7 中生效，所以还是 hack 一下比较合适。至此产生了我们熟悉的兼容各个浏览器的 inline-block 写法。  小结：IE6、7 并不是不支持 inline-block，只是 block 元素需要做一些处理来达到 inline-block 的效果。  2. 到底什么是 inline-block  说了很多，或许很多朋友还不是太明白到底什么是 inline-block？W3C 在 CSS2.1 The ‘display’ property 中描述如下：  This value causes an element to generate an inline-level block container. The inside of an inline-block is formatted as a block box, and the element itself is formatted as an atomic inline-level box.  大致意思就是：inline-block 后的元素创建了一个行级的块容器，该元素内部（内容）被格式化成一个块元素，同时元素本身则被格式化成一个行内元素。  直白一点的意思就是：inline-block 的元素既具有 block 元素可以设置宽高的特性，同时又具有 inline 元素默认不换行的特性。当然不仅仅是这些特性，比如 inline-block 元素也可以设置 vertical-align 属性。简而言之：  inline-block 后的元素就是一个格式化为行内元素的块容器( Block container )  怎么样？听起来还不错吧！  3. inline-block 缘从何起？  前面已经证明了 IE 5.5 开始就支持了 inline-block，那么 IE5.5 是什么时候发布的呢？话说当年网景与 IE 大战，IE5.5 那是何等的风骚……（好吧，此处略去十页）。从维基百科的资料来看，IE5.5 beta1 的发布时间是：1999年12月，最终版本是 2000年7月。那么 W3C 标准中是何时才出现 inline-block 这个值的呢？  在 CSS1 规范中，「display」的值仅包括： block | inline | list-item | none 。CSS2.1中才添加了 inline-block 属性值。一丝继续舔着手指，用那苦逼的英语水平终于翻到了这份草案：[http://www.w3.org/TR/2002/WD-CSS21-20020802/visuren.html#display-prop，](http://www.w3.org/TR/2002/WD-CSS21-20020802/visuren.html" \l "display-prop，" \t "http://forsigner.com/2013/03/28/inline-block/_blank) 这份草案的日期是 2002年8月2日，纳尼！！！原来我们纠结了半天的 inline-block ， IE5.5 至少提前两年就提出来了啊！难道是微软给 W3C 提议后，CSS 2.1才加入的？（不过我看到 W3C 官网有一个关于是否增加 inline-block 的投票）好吧这个问题也许有一天 IE 某个开发者写《 IE回忆录》的时候我们才能了解到其中的内幕。如果找到更早关于 inline-block 的 CSS草案，也麻烦告知一丝一声。好吧，如果你还不相信，打微软官方电话问问吧 800-820-3800（不是 DHC 哦！）。  原来我们一直讨论的 inline-block 在 IE 6、7中和 CSS2.1 中的（现代浏览器所支持的） inline-block 上压根不是一个东东嘛，IE6、7 中的 inline-block 更像是 IE 的私有属性值，他们本身就不具有可比性。简单、绝对的认为 IE6、7 不支持 inline-block 好比一叶障目，看到前面，却看不到后面，太过于片面。诚然，IE6、7 的 hasLayout 给我们带来了很多麻烦，但是不得不承认微软的 IE 在网页多语言文本混排上的先进性，尤其是 CJK 文字和西文的混排，超越其他浏览器至少5年。  总结：  IE5.5 后开始支持 inline-block， 但是它所支持的 inline-block 不能等同于 CSS2.1 中的 inline-block，因为 IE5.5 比 CSS2.1 更早提出 inline-block 的概念并作为所谓的「私有属性值」使用，所以二者表现出来的效果是不完全一致。  IE 5.5、6、7 、8（Q）中 block 元素对 inline-block 支持不完整，如果要达到类似的效果，需要先设置为 display:inline，然后使用 zoom:1 等触发 hasLayout。  IE 5.5、6、7 、8（Q）中 inline 元素欲达到 inline-block 的效果只需直接设置此属性值或使用 zoom:1 等均可。  各浏览器对 display 属性的支持情况请参阅：《各浏览器对 ‘display’ 特性值的支持程度不同》 |
| inline-block 今生 | 1. display:inline-block 后的元素为什么会产生水平空隙，这真的是 bug 吗？  这么一个神奇的属性，为何大家一直避而远之呢？这恐怕还得从 inline-block 元素之间产生的水平空隙（间隙）说起吧。  现代浏览器中 inline 和 block 元素 display:inline-block 后均会产生水平空隙；  IE6、7，IE8（Q）模拟 display:inline-block 后分两种情况：  IE6、7，IE8（Q）中：inline 元素会产生空隙，block 元素不会产生空隙。 看看 inline 元素默认的表现情况如何？原来默认就有空隙存在！它们是谁？是空白符（white space）！  W3C 9.1 White space 中规定以下元素属于空白符（white space）：  ASCII 空格  ASCII 制表符  ASCII 换页符  零宽度空格 「这个在闭合浮动中也有运用到」  9.3.2 Controlling line breaks 中进一步阐述：  A line break is defined to be a carriage return ( ), a line feed ( ), or a carriage return/line feed pair. All line breaks constitute white space.  For more information about SGML’s specification of line breaks, please consult the notes on line breaks in the appendix.  折行被定义为一个回车符（ ），一个换行符 line feed ( )，或者一个回车、换行的组合。所有的折行构成了空白符。  有关 SGML 规范中折行的更多信息，请参阅附录中关于折行的注释。  通常情况下，对于多个连续的空白符（空格，换行符，回车符等），浏览器会将他们合并为一个空白符。CSS 中由 white-space 这个属性来控制：  white-space：normal | pre | nowrap | pre-wrap | pre-line  默认值：normal  normal：默认处理方式。  pre：用等宽字体显示预先格式化的文本，不合并文字间的空白距离，当文字超出边界时不换行。可查阅 pre 对象  nowrap：强制在同一行内显示所有文本，直到文本结束或者遭遇 br 对象。  pre-wrap：用等宽字体显示预先格式化的文本，不合并文字间的空白距离，当文字碰到边界时发生换行。  pre-line：保持文本的换行，不保留文字间的空白距离，当文字碰到边界时发生换行。  注：IE7及更早浏览器不支持 CSS2.1 新增的 pre-wrap | pre-line。  所以这并不是 inline-block 后产生的 bug，而是因为 inline-block 具有 inline 元素固有的特性。那么为何 IE6、7 block 元素没有产生空隙呢？其实前面也提到了 IE 的 hasLayout，具有独立性，所以产生 hasLayout 的元素之间表现出来互不影响，这也再次表明 IE6、7 中的 inline-block 不能等同于 CSS2.1 中的 inline-block。如果非要说是有 bug， IE6、7 block 元素 inline-block 后不产生空隙才是 bug。  测试表明删除换行符后，inline 元素间的空隙就「消失」了：  2.去掉 inline-block 产生的空隙  为了让各个浏览器表现一致，更好的还原视觉设计搞，很多时候我们需要去掉 inline-block 产生的空隙。  上一节中我们已经知道产生空隙的根本性原因是：  HTML 中的换行符、空格符、制表符等产生了空白符，而这些归根结底都是字符，那么它们的大小都是 受 font-size 来控制的，字体大小直接导致 inline 或者 inline-block 后元素之间空隙的大小，把 inline-block 元素间的空隙认为总是某个固定大小是错误的。  用 GIF 动画的形式来表明对应关系：  很清楚的看到，当 font-size:0 的时候元素间的空隙都为0了，或许到这里你会感到很欣喜了，原来掌握的根本性原因这么简单就搞定了啊！  然，理想是丰满的，现实是骨感的。  大部分浏览器是支持 font-size:0 的。很明显，我们要和 IE 6、7 这两个妖孽进行一番战斗。  font-size:0 的支持情况  1）Chrome  低版本的 chrome 浏览器为了不让文字过小不利于阅读，默认是不支持 font-size:0 的，还好我们有 -webkit-text-size-adjust 这个私有属性来控制，当设为 none 时就支持字体大小为 0 了。我已经记不清楚 chrome 从哪个版本开始支持 font-size:0 了，反正我用 chrome 19 是支持了（有知道的朋友，烦请告诉一丝一声，最好有官方更新说明）。但是，-webkit-text-size-adjust:none; 会直接导致页面文字无法缩放，这对于用户来说显然是不友好的。所以-webkit-text-size-adjust:none; 一定要慎用，确保使用的地方没有大面积的文字。  -webkit-text-size-adjust:none 的使用场景实例参阅：[http://vip.etao.com/](http://vip.etao.com/" \t "http://forsigner.com/2013/03/28/inline-block/_blank)  2）Safari  Safari 5 依旧不支持 font-size:0 ，不过相信这些浏览器厂商都意识到了这个问题，在 Mac 平台最新的 Safari 6 已经很好的支持 font-size:0 了。  3）Firefox，Opera  经测试，Firefox12，Opera 10 ，这次表现不错，支持 font-size:0 。  4）IE  IE8 以上支持 font-size:0; IE6、7 inline 元素 inline-block 后设置 font-size:0 始终有 1px 的空隙。 是不是一下子又开始头疼了？没关系，让我们请出 letter-spacing 和 word-spacing 二位大神。既然空白符也是字符，那么二位大神肯定是可以搞定它们的。  letter-spacing : normal | length （检索或设置对象中的文字之间的间隔）  word-spacing : normal | length（检索或设置对象中的单词之间插入的空隔）  normal： 默认间隔  length： 用长度值指定间隔，允许为负值。  还等什么，我们赶紧试试吧：  参照 DEMO  第一步：使用 font-size:0经测试发现，chrome、firefox、IE8+、opera，inline 或 block 元素都没有空隙了； Safari 5.1.7 由于不支持 font-size:0 ，仍然存在空隙；  IE6、7、8（Q），inline 元素 inline-block 后始终存在 1px 左右的空隙。 第二步：处理 Safari 不支持 font-size:0 的问题上面已经指出 letter-spacing 是支持负值的，那么这个负值到底取多少合适呢？经过测试得出的结论是：inline-block 产生的空隙与父级元素继承或者设定的 font-family、font-size 有关，通常情况下，12px 大小的 tahoma 字体，inline-block 后元素间产生的空隙（间隙）大约是 5px； 各个字体详细情况请参阅《inline-block空隙–letter-spacing与字体大小/字体关系数据表》。  Firefox 中 letter-spacing 负值的绝对值大于空隙大小后，会导致元素整体位置向右偏移；  Safari 中 letter-spacing 负值的绝对值大于空隙大小后，内部会发生重叠。  第三步：修复 IE6、7 中始终存在的 1px 空隙 既然 letter-spacing 已经无能为力了，那就试试 word-spacing 吧，直接设置 word-spacing:-1px。这里需要注意的是，letter-spacing 和 word-spacing 同时使用可能导致冲突，所以我们需要在 IE6、7 中 hack 掉 letter-spacing。最终代码如下：   |  | | --- | | font-size:0;/\* 所有浏览器 \*/ letter-spacing:-5px;/\* Safari 等不支持字体大小为 0 的浏览器 \*/ \*letter-spacing:normal; word-spacing:-1px;/\* IE6、7 \*/ |   第四步：子元素重置回正常值 上述所有操作都是在父元素设置的，那么子元素都会继承这些属性，字体大小为0了，子元素就什么都看不到了，这并不是我们想要的。 同时字符和单词间距我们也要把它重置为默认值。「font-size: 12px; letter-spacing: normal; word-spacing: normal;」 最后：inline-block 更好的复用 或许你会担心每次我都要去看字体和空隙之间大小的关系吗？其实不然，通常情况下，全局字体都已经在 body 中指定了，根据全局字体设置合适的 letter-spacing 负值即可。如此一来，我们便可以放心大胆的使用 inline-block 了，结合 OOCSS 的思想，可以抽离出两个复用的类，在需要设置 inline-block 元素的父级元素上定义一个「.dib-wrap」，该元素自身定义为「.dib」。这里还有一个问题需要注意的是：由于 inline-block 具有 inline 元素的特性，在垂直方向上很多时候我们并不希望元素以「vertical-align:baseline」方式来呈现，所以在「.dib-wrap」中统一重置为「vertical-align:top」即可。  3. 去除 inline-block 空隙终极解决方案（2012年8月17日更新）   |  | | --- | | .dib-wrap {  font-size:0;/\* 所有浏览器 \*/  \*word-spacing:-1px;/\* IE6、7 \*/ } .dib-wrap .dib{  font-size: 12px;  letter-spacing: normal;  word-spacing: normal;  vertical-align:top; } @media screen and (-webkit-min-device-pixel-ratio:0){ /\* firefox 中 letter-spacing 会导致脱离普通流的元素水平位移 \*/ .dib-wrap{  letter-spacing:-5px;/\* Safari 等不支持字体大小为 0 的浏览器, N 根据父级字体调节\*/ } } .dib {  display: inline-block;  \*display:inline;  \*zoom:1; } |   其实在 YUI 3 中也全面运用了 inline-block 作为基础布局，YUI 3 是这样解决的：   |  | | --- | | .yui3-g {  letter-spacing: -0.31em; /\* webkit: collapse white-space between units \*/  \*letter-spacing: normal; /\* reset IE < 8 \*/  word-spacing: -0.43em; /\* IE < 8 && gecko: collapse white-space between units \*/ }  .yui3-u {  display: inline-block;  zoom: 1; \*display: inline; /\* IE < 8: fake inline-block \*/  letter-spacing: normal;  word-spacing: normal;  vertical-align: top; } |   显然，这里纯粹使用了 letter-spacing 和 word-spacing 来控制元素间的空隙，局限性极大，-0.31em 和 -0.43em 只是因为 YUI 3 全局 css fonts.css 里设置是：「body { font:13px/1.231 arial,helvetica,clean,sans-serif; }」。  当然，如果你坚持使用把 html 写在一行的方式来达到去除 inline-block 空隙的目的，我只能说：一切以牺牲结构来兼容表现的行为都是耍流氓！所以探讨此种方式去除空隙也将是无意义的，不在本文和作者考虑范围之内。  4. 结局——本文产生的一些观点如下：  IE5.5 后开始支持 inline-block， 比 CSS2.1 更早提出 inline-block 的概念并作为所谓的「私有属性值」使用。但是它所支持的 inline-block 不能等同于 CSS2.1 中的 inline-block，IE 5.5、6、7 、8（Q）中 block 元素对 inline-block 支持不完整，因此二者表现出来的效果是不完全一致。 产生 inline-block 空隙的根本性原因是：HTML 中的换行符、空格符、制表符等合并为空白符，字体大小不为 0 的情况下，空白符自然占据一定的宽度，因此产生了元素间的空隙。 慎用 -webkit-text-size-adjust:none，它将会导致页面无法通过缩放来改变字体大小。  三、inline-block 未来  如今，Mac 平台下的 Safari 6 已经支持 font-size:0 了，相信很快 Windows 平台的 Safari 如果发布 5.X 的更新，也会支持字体为 font-size:0 了。等到 IE6、7 灭亡之后，世界就真真儿的美妙了！最后说一点：inline-block 与 float 也是无法直接比较的，请不要再讨论 inline-block 和 float 哪个更好的话题了。inline-block 从 IE5.5 一路走来，存在即是合理，以后有时间在总结一下 inline-block 与 float 的使用场景的区别。 |

float与 inline-block

|  |  |
| --- | --- |
| www | <http://www.zhangxinxu.com/wordpress/2010/11/%E6%8B%9C%E6%8B%9C%E4%BA%86%E6%B5%AE%E5%8A%A8%E5%B8%83%E5%B1%80-%E5%9F%BA%E4%BA%8Edisplayinli>ne-block%E7%9A%84%E5%88%97%E8%A1%A8%E5%B8%83%E5%B1%80/ |
| float的使用局限性 | 对于浮动局部的局限性，想必同行们都知道，就是每个列表元素的高度必须要一致，否则就会像是俄罗斯方块一样，"锯齿相错"，例如一个左浮动列表布局，如果第一行有个列表高度高于其他列表，那就在第二行，第一个元素会沿着最高元素的右侧对齐，此原因是属于恶魔系的float属性破坏了line级height  <**style**>  .floatOut {  border: 1px solid lightseagreen;  height: 500px;  width: 500px;  }  .floatOut **div** {  float: left;  width: 110px;  }  .a { height: 100px; } .b { height: 200px; } .c { height: 300px; } .d { height: 100px; } .e { height: 100px; } </**style**>  <**div** class="floatOut">  <**div** class="a">1</**div**>  <**div** class="b">2</**div**>  <**div** class="c">3</**div**>  <**div** class="d">4</**div**>  <**div** class="e">5</**div**>  <**div** class="e">6</**div**>  <**div** class="e">7</**div**> </**div**> |
| inline-block的使用 | <**style**>  .floatOut {  border: 1px solid lightseagreen;  height: 500px;  width: 500px;  }  .floatOut **div** {  display: inline-block;  width: 110px;  }  .a { height: 100px; } .b { height: 200px; } .c { height: 300px; } .d { height: 100px; } .e { height: 100px; } </**style**>  <**div** class="floatOut">  <**div** class="a">1</**div**>  <**div** class="b">2</**div**>  <**div** class="c">3</**div**>  <**div** class="d">4</**div**>  <**div** class="e">5</**div**>  <**div** class="e">6</**div**>  <**div** class="e">7</**div**> </**div**> |
| 空格 | 空格符本质上就是个字符，与a,b,c,d这些字符是个同一个属性的东西，只是他是空格，透明的看不见而已（但可以选中）。  font-size:0;  Firefox浏览器：正常  IE7浏览器：1像素的空格  Chrome浏览器：无效  letter-spaceing:-4px;  Firefox3.6浏览器：正常  Chrome浏览器：正常  IE7浏览器：正常  结论： » block水平的元素inline-block化后，IE6/7没有换行符间隙问题，其他浏览器均有； » inline水平的元素inline-block后，所有主流浏览器都有换行符/空格间隙问题； » font-size:0，去除换行符间隙，在IE6/7下残留1像素间隙，Chrome浏览器无效，其他浏览器都完美去除； » letter-spacing负值可以去除所有浏览器的换行符间隙，但是，Opera浏览器下极限是间隙1像素，0像素会反弹，换行符间隙还原。  应用display:inline-block属性实现列表布局的几个关键字就是：block水平的标签，font-size:0和letter-spacing负值。 |
| 注意 | 使用white-space:nowrap;属性可以让列表不换行；可用在列表元素的水平滚动切换  使用vertical-align:top;属性可以控制列表上下对齐方式；  使用text-align:justify;可以实现自动等宽水平排列的列表布局，而且是两端对齐的，不需要计算宽度，一切都是浏览器自动的，很方便很强大。  http://www.zhangxinxu.com/study/201011/inline-block-text-justify-layout.html  使用text-align: center、left、right;可以控制其对齐方式； |

inline-block/text-align:justify下列表的两端对齐布局

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | <http://www.zhangxinxu.com/wordpress/2011/03/displayinline-blocktext-alignjustify%E4%B8%8B%E5%88%97%E8%A1%A8%E7%9A%84%E4%B8%A4%E7%AB%AF%E5%AF%B9%E9%BD%90%E5%B8%83%E5%B1%80/> | | | |
| 简介 | CSS2中text-align有一个属性值为justify，为对齐之意。其实现的效果就是可以让一行文字两端对齐显示（文字内容要超过一行）。  text-align其诞生的意义是控制文字的对齐与显示的，从其属性名上就可以看出来。从其渲染与解析上来看，其主要是用来控制inline水平元素或inline-block元素的对齐与显示的，例如嵌套行内标签的文字、图片、input表单控件等；而对block水平的元素是没有作用的。  所以，对于列表元素，理论上，我们只要将原本block水平的列表元素inline化或是inline-block化就可以轻松实现其两端对齐了。然而考虑到实际情况，inline水平化显然是不可能的，因为不能给列表元素定宽定高，设置垂直方向上的间距等，列表元素就像是一摊烂泥，根本没法用来砌房子；而inline-block化也是有重重阻碍的，因为IE6/7并不真正意义上的支持inline-block属性。 | | | |
| 1、外层元素定宽 | <**style**>  .out{  width: 500px;  height: 800px;  }  .out>**div**{  float:left;  width: 100px;  height: 100px;   margin: 10px;  text-align: center;  }  </**style**>  <**div** class="out">  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**> </**div**> | 需要精确计算外层div在一行几个子元素的最小宽度；否则会出现大空缺的情况 | |
| 2、margin扩大外层元素宽度 | 人人网这里的列表布局为inline-block布局。  人人网热门分享inline-block布局 张鑫旭-鑫空间-鑫生活  其通过也是通过增大父标签的宽度来实现看上去的"两端对齐"效果的，不过其不是通过width属性来增加父标签的宽度的，而是使用的margin负值（我个人推荐使用margin负值而不是定宽）：  margin负值增加父容器的宽度 张鑫旭-鑫空间-鑫生活 | | |
| 3、nowrap单行列表元素两端对齐，且不换行 | <**style**>  .out{  display: inline-block;  height: 800px;  white-space: nowrap;  **/\*一般用在实现单行列表元素看上去的"两端对齐"效果上\*/** }  .out>**div**{  display: inline-block;**/\*子布局\*/** width: 100px;  height: 100px;  margin: 10px;  text-align: center;  } </**style**>  <**div** class="out">  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**> </**div**> | | 利用white-space: nowrap，此方法需在inline-block布局基础上使用，一般用在实现单行列表元素看上去的"两端对齐"效果上。white-space: nowrap会强制列表元素不换行，于是你无需设定父标签容器的宽度或是通过margin负值等手段增加父容器的宽度等。这里不展开，以后有机会要好好说一说white-space: nowrap这个很有用的CSS声明的。 |
|  | 以上就是目前几种常见的实现列表元素看上去的"两端对齐"效果方法。怎么样，是不是每个都很折腾——首先列表元素排列就已经很折腾人的了（定宽，计算间距），然后还有人为增加父容器的宽度，同时祖辈元素还要溢出隐藏(overflow:hidden)，oh，my lady嘎嘎。  我想这就是为什么网上会有前端工程师咆哮体咆哮加班的原因了（关于此图，您可以点击[这里](http://ww4.sinaimg.cn/bmiddle/41718533jw6df4iuu2d6qj.jpg" \t "http://www.zhangxinxu.com/wordpress/2011/03/displayinline-blocktext-alignjustify%e4%b8%8b%e5%88%97%e8%a1%a8%e7%9a%84%e4%b8%a4%e7%ab%af%e5%af%b9%e9%bd%90%e5%b8%83%e5%b1%80/_blank)查看）。 | | |
| 4、text-align:justify实现两端对齐 | text-jusyify下的inline-block自适应列表布局实例页面 | | |
| <**style**>  .box {  width: 50%;  padding: 20px;  margin: 20px auto;  background-color: #f0f3f9;  text-align: justify;  }   .list {  width: 120px;  display: inline-block;  padding-bottom: 20px;  text-align: center;  vertical-align: top;  } </**style**>  <**div** class="box">  <**span** class="list"><**img** src="http://image.zhangxinxu.com/image/study/s/s128/mm9.jpg" /> 哇哦，美女，口水，鼻血~~~</**span**>  <**span** class="list"><**img** src="http://image.zhangxinxu.com/image/study/s/s128/mm9.jpg" /> 哇哦，美女，口水，鼻血，不行了，我的小兔乱撞~~</**span**>  .  .  . </**div**> | | |
| 以ul与li来说明代码书写的注意事项  核心代码：  ul{text-align:justify;}  li{display:inline-block;}  注意点：  列表元素首尾标签留空（或换行），不能够上一个标签组的结束标签与下一个标签组的其实标签连在一起    对于IE8浏览器，列表元素不能处在font-size:0的环境下，至少code>font-size:1px，因为IE8浏览器font-size:0或直接把换行空格或普通空格抹掉而无法实现两端对齐效果。  显然上面的ul, li样式组合在IE6/7浏览器下是行不通的，即使你使用hack让IE6/7下的li标签有类似于display:inline-block的特性也是没有作用的。那么如何才能让IE6/7浏览器也有列表元素支持text-align:justify属性呢？经过我反复试验与调试，总结了两点：inline标签化以及结束标签连续化。  1. inline标签化：就是列表元素需要使用inline水平的标签，例如span, a, strong, em等，像li, div这些标签就不可以。  2. 结束标签连续化：是指列表元素及其内部标签的结束标签需要连在一起。  =>  我们已经习惯了结构化的缩进，所以上面结束标签连写看上去很不自然，有些别扭。但是，为了实现效果，这是没有办法的事情。注意：如果列表标签内嵌多层，则所有层级的结束标签都要连续。  IE6/IE7浏览器同时满足上面的inline标签化以及结束标签连续化，再加上先前现代浏览器下的首尾标签留空，IE6/IE7浏览器也就能够实现列表元素的两端对齐啦！  为了便于更直观的知道各个浏览器下实现两端对齐效果需要注意的事项，我特地制作了下表：   |  |  | | --- | --- | | 各个浏览器实现text-align:justify下的两端对齐布局注意事项表 | | | 浏览器 | 注意事项 | | IE6 | inline水平列表标签、列表结束标签连续、列表元素间换行或留空 | | IE7 | inline水平列表标签、列表结束标签连续、列表元素间换行或留空 | | IE8 | 列表元素间换行或留空、列表元素的环境字体大小不能为0 | | 现代浏览器 | 列表元素间换行或留空 | | | |
| <**style**>  .out{  height: 800px;  width: 600px;  text-align: justify;  **/\*white-space: nowrap;\*//\*一般用在实现单行列表元素看上去的"两端对齐"效果上\*/** }  .out>**div**{  display: inline-block;**/\*子布局\*/** width: 100px;  height: 100px;  margin: 10px;  text-align: center;  } </**style**>  <**div** class="out">  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  --- </**div**>  还有一个很悲剧的问题没有解决，就是当列表元素最后一行无法两端对齐:  列表（或文字）要两端对齐的前提就是内容必须超过一行，所以，要解决最后一行元素无法两端对齐的文字其实很简单，就是在列表（或文字段）的最后创建一个高度为0的宽度100%的透明的inline-block的标签层就可以了，例如：  .justify\_fix{display:inline-block; width:100%; height:0; overflow:hidden;}  <span class="justify\_fix"></span>  => | | |
|  | 很多时候，我们希望列表的最后一行是左对齐排列的，而不是两端对齐，这时候怎么办呢？原理与上面的两端对齐一致。就是复制几个列表元素的外层标签，等宽，但高度为0，里面就是个&nbsp;(不可缺)，复制的个数一般就是每行元素的列表个数啦，这样肯定可以保证最后一行元素一定是左对齐排列的啦！  .left\_fix{height:0; padding:0; overflow:hidden;}  <span class="list left\_fix">&nbsp;</span>  在margin的影响下，存在一定问题，建议inline-block元素之间的间距通过padding来实现  <**style**>  .out{  width: 600px;  text-align: justify;  **/\*white-space: nowrap;\*//\*一般用在实现单行列表元素看上去的"两端对齐"效果上\*/** line-height: 0;**/\*+++\*/** }  .out>**div**{  line-height: normal;**/\*+++\*/** display: inline-block;**/\*子布局\*/** width: 100px;  height: 100px;  text-align: center;  margin: 10px;**/\*影响+++\*/** }   .justify\_fix {  display: inline-block;  width: 100%;  height: 0;  overflow: hidden;  }   .out>.left\_fix {  height: 0;  padding: 0;  **/\*margin: 0;\*//\*影响未知\*/** overflow: hidden;   vertical-align: top;**/\*+++\*/** }  </**style**>  <**div** class="out">  <**div**>  <**img** src="images/3.jpg" width="80"/>  <**div**>刘亦菲</**div**>  </**div**>  ---  <**div** class="list left\_fix">&nbsp;</**div**>  --- </**div**> | | |
| demo | 我们拿上面人人网热门分享处的inline-block列表布局举例。  您可以狠狠地点击这里：[人人网热门分享列表text-justify下两端对齐demo](http://www.zhangxinxu.com/study/201103/inline-block-text-justify-renren.html" \t "http://www.zhangxinxu.com/wordpress/2011/03/displayinline-blocktext-alignjustify%e4%b8%8b%e5%88%97%e8%a1%a8%e7%9a%84%e4%b8%a4%e7%ab%af%e5%af%b9%e9%bd%90%e5%b8%83%e5%b1%80/_blank)  效果如下图（截自IE7浏览器）：  人人网热门列表justify下两端对齐效果截图 张鑫旭-鑫空间-鑫生活  CSS代码如下：  <**style**>  .video-list {  **/\*列表父容器\*/** width: 540px;  margin-left: auto;  margin-right: auto;  text-align: justify;  }  .text-justify-list {  **/\*列表元素\*/** display: inline-block;  width: 97px;  margin-bottom: 15px;  text-align: left;  vertical-align: top;  }  .justify\_fix {  **/\*末行悲剧变喜剧\*/** display: inline-block;  width: 100%;  height: 0;  overflow: hidden;  } </**style**>  可以看到列表元素压根就没有设置垂直方向上的margin或是padding值，就单单一个宽度值，但是列表元素确实两端对齐，等间距排列。没有计算，没有有意去增加父容器宽度等，超简单就实现了。  比对上面提到的些注意事项，看看这个例子中的HTML代码是如何实践上面的注意事项的：  ①inline水平标签  如下截图所示： inline标签使用截图 张鑫旭-鑫空间-鑫生活  ②列表结束标签连续  如下图所示： 结束标签连续 张鑫旭-鑫空间-鑫生活  ③列表标签换行或留空  如下截图所示： 列表标签换行不连续截图 张鑫旭-鑫空间-鑫生活  如此，你也可以轻轻松松实现列表元素的两端对齐布局，而且不用去担心兼容性问题！！GO！大胆的去使用吧！ | | |
| inline-block瀑布流布局 | <**style**>  #container{  **/\*width: 600px;\*//\*body\*/** }  .column {  display: inline-block;  vertical-align: top;  width: 200px;  }  #column0{  width: 600px;  }  #column1{  width: 400px;  }  #column2{  **/\*width: 400px;\*/** }   #column3{  **/\*width: 400px;\*/** }  .column-li {  display: block;  padding: 5px;  border: 1px solid #ccc;   text-decoration: none;  text-align: center;  }  .column-li **img** {  display: block;**/\*将图片与文字隔开的方式：其中一个设置为block\*/** margin: 0 auto 5px;  border: 0;  vertical-align: bottom;  }  .column-li **strong** {  color: #333;  } </**style**> <**div** id="container">   <**span** id="column0" class="column">  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_003.jpg"> <**strong**>004 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_004.jpg"> <**strong**>005 </**strong**> </**a**>  </**span**>  <**span** id="column1" class="column">  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_000.jpg"> <**strong**>001 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_001.jpg"> <**strong**>002 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_002.jpg"> <**strong**>003 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_003.jpg"> <**strong**>004 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_004.jpg"> <**strong**>005 </**strong**> </**a**>  </**span**>  <**span** id="column2" class="column" >  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_009.jpg"> <**strong**>006 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_011.jpg"> <**strong**>007 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_022.jpg"> <**strong**>008 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_003.jpg"> <**strong**>009 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_004.jpg"> <**strong**>010 </**strong**> </**a**>  </**span**>  <**span** id="column3" class="column">  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_008.jpg"> <**strong**>0011 </**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_007.jpg"> <**strong**>0012</**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_008.jpg"> <**strong**>0013</**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_006.jpg"> <**strong**>0014</**strong**> </**a**>  <**a** href="###" class="column-li"> <**img** src="http://cued.xunlei.com/demos/publ/img/P\_005.jpg"> <**strong**>0015</**strong**> </**a**>  </**span**> </**div**> | | |

line-height

line-height简介

|  |  |  |
| --- | --- | --- |
| 定义 | 1）line-height和font-size 有联系吗，什么联系？  2）line-height的继承特性有什么特殊性？  如果你对这2个问题很清楚的话，你可以跳过本文。  首先看看w3c的解释：line-height 属性设置行间的距离（行高，行间距）。  该属性会影响行框的布局。在应用到一个块级元素时，它定义了该元素中基线之间的最小距离而不是最大距离。line-height 与 font-size 的计算值之差（在 CSS 中成为"行间距"）分为两半，分别加到一个文本行内容的顶部和底部。可以包含这些内容的最小框就是行框。原始数字值指定了一个缩放因子，后代元素会继承这个缩放因子而不是计算值。什么是基线（baseline）？在我之前的一篇文章中已经给出了一个地址，如果不太清楚的话，可以去看看。一个重要的特性就是line-height可以被继承。  浏览器的默认值不一样  各个浏览器的默认值不一样（在1~1.2），所以我们看到的行高都不一样，也是由于这个原因，一般网站都有reset。  1:行内元素设置line-height时，不论设置的多大，它的背景永远只跟随文字的大小来渲染，但是行高的大小还是会体现的。  <span style="background-color:red; line-height:30px;">行高对背景的渲染不起作用</span>  2:行内块和块元素，背景颜色会随着line-height的设置来给文字背景加相应的颜色，line-height设置多少会直接影响高度的。  <p style="background-color:red; line-height:30px;">行高对背景的渲染起作用</p> | |
| 示意图 |  | |
|  | <http://www.jb51.net/css/199172.html> | |
| 基础概念 | 1. 四线   基线（Base line），指的是一行字横排时下沿的基础线，基线并不是汉字的下端沿，而是英文字母x的下 端沿，同时还有文字的顶线（Top line）、中线（Middle line）和底线（Bottom line），用以确定文字行的位置。     1. 行高   行高指的是文本行的基线间的距离，但是文本之间的空白距离不仅仅是行高决定的，同时也受字号的影响。   1. 行距   行距是指行高与字体尺寸的差(leading)     1. 内容区域   一行中的每个元素都有一个内容区域，它是由字体尺寸决定的。     1. 行内框   行内元素会生成一个行内框（inline box），行内框只是一个概念，它无法显示出来，但是它又确实存在；  在没有其他因素影响的时候，行内框等于内容区域；  设定行高则可以增加或者减少行内框的高度，即：将行距的值（行高-字体尺寸）除以2，分别增加到内容区域的上下两边     1. 行框   由于行高可以应用在任何元素上，因此同一行内的若干元素可能有不同的行高和行内框高:  <p style="line-height:20px;">行高20px。<strong style="line-height:50px;"> 行高50px。</strong><span style="line-height:30px;">行高30px。</span></p>  行框（line box）同行内框类似，是指本行的一个虚拟的矩形框，其高度等于本行内所有元素中行高最大的值。因此，当有多行内容时，每行都会有自己的行框， | |
| 行高的计算与继承 | 以em、ex和百分比为单位的行高，其基数是元素本身的字体尺寸。  例如有代码如下：  <p style="font-size:20px;line-height:2em;">字高20px，行高2em。</p>  <p style="font-size:30px;line-height:2em;">字高30px，行高2em。</p>  2个段落的行高都为2em，但是字体大小不同，因此显示如图7-23所示。  IMG_256 | |
|  | 行高可以设定得比字体高度小，此时多行的文字将叠加到一起，例如有如下代码，其显示如图7-24所示。  p { font-size : 20px; line-height :10px; }  <p>字高20px，行高10px。此时多行的文字将叠加到一起。</p>  IMG_256 | |
| 行高是可继承的，但是继承的是计算值，例如有如下代码：  <style>  p {  font-size: 20px;  line-height: 2em;  }  p span {  font-size: 30px;  }  </style>  <p>字高20px。<span>字高30px。</span></p>  <p>元素的行高2em，字体尺寸为20px，因此计算值为40px，虽然<span>元素本身的字体尺寸为30px，不过其继承的行高仍为40px。但是在不同的浏览器内显示的效果却不尽相同，如下  IMG_256 | |
| 由于继承的是计算值，因此当元素内的文字字体尺寸不一样的时候，如果设定固定的行高很可能造成字体的重叠，例如有如下代码，其显示如下  p { font-size : 20px; line-height : 1em; }  p span { font-size : 30px; }  <p>字高20px，行高1em，当文本为多行时可能会发生文字重叠的现象。<span>字高30px。</span></p>  IMG_256  为了避免这种情况，可以为每个元素单独定义行高，但是这样很烦琐，因此可以定义一个没有单位的实数值作为缩放因子来统一控制行高，缩放因子是直接继承的，而不是继承计算值。例如修改上例中的行高为：  p { line-height : 1; }  IMG_256  当内容中含有图片的时候，如果图片的高度大于行高，则含有图片行的行框将被撑开到图片的高度:  IMG_257   注意：图片虽然撑开了行框，但是不会影响行高，因此也不会影响到基于行高来计算的其他属性。  提示：当行内含有图片的时候，图片和文字的垂直对齐方式默认是基线对齐，关于垂直对齐将在本章[7.4 垂直对齐：vertical-align属性]一节中讨论。 | |
| +单位 计算 继承 | 行高的几种表示方法：px/em，或normal，或百分值，或数值，或inherit继承。  1）div{line-height：nomal}就是默认的形式，不太推荐这种方式，因为各个浏览器都不一样，会有差异，firefox大概是34px。oprea大概是30px。  2）div{line-height:inherit}继承的方式，就是继承父级元素的形式，不定义的默认值，也不推荐这种方式。  3）div{line-height:150%} 百分比的形式，比较的灵活。  4）div{line-height：20px} 长度的形式，关于常见的一些CSS中的长度单位，后面我将讲述一下。  5）div{line-height:1.5} 纯数字的方式，我推荐的方式，能适应各种。  在显示文章的box里，px的表示方法首先是要被淘汰的。因为文章里面的文字是有大有小的，使用px定值，由于继承性，无法实现根据文字大小自动调整间距，会出现大号文字重叠的现象。  normal也是不行的，一般文章显示最好是650像素的宽度，1.5倍的行距较好。一般浏览器的normal值在1~1.2之间，使用normal必然文字间距过小，阅读吃力。  百分值也有继承性，但是有个很搓的办法可以实现文字间距自动适应于文字的大小，那就是使用"\*"通配符，例如：.article\_box \*{line-height:150%;}就不会出现文字重叠的情况了;使用"\*"通配符大大增加了css的渲染，效率低  更好的方法就是使用数值。150%虽然和1.5在值上是一样的，但是它们也是有差别的，差别在于继承性，使用百分比会计算line-height的值，然后以px像素为单位继承下去，而1.5则是先继承1.5这个值，遍历到了该标签再计算去line-height的像素值。所以同样的效果只需要.article\_box{line-height:1.5;}就可以实现了。 | |
| 关于继承特性  1）百分比的继承：继承下来的元素会忽略font-size的大小，都是用同一值。如下图所示：  IMG_256  2）长度的继承：长度是一个绝对值，同样的，继承下来的元素会忽略本身的font-size的大小的。这个可以将上面的例子150%换成18px，效果和上面一样。  3）默认值的继承：这里和前面的方式有点不一样，后面的元素不会忽略font-size的大小，而是基于font-size的大小来计算高度的。我是用firefox3.6测试的，各个元素的font-size的大小不一样，基本上各行文字不再重叠在一起了。后来我用firebug看了下各行的高度，h4是40px，#test是54px（由于是在2行排列的），但是p是35px，我用了chrome10测试了下，h4是40px，#test是46px，p是30px。这2个浏览器就表现不一样了。可见这个方式不是一个好的方式，会带来浏览器的差异。  4）数字的方式：这是对继承方式的一个加强，后面的元素会根据font-size的大小来计算高度的，系数就是这个数字，可见这个在所有的浏览器中都是一样的，可以将上例子中的150%换成1.5试试。 | |
| x浏览器的差别与错误 | 浏览器在显示的时候往往会有自己的表现形式，例如在Opera内，行高将按照CSS定义的将行距除以2增加到内容区域的上下两边，而IE和Firefox则不是完全平分，如图7-29所示。  IMG_256 图7-29 不同浏览器对行高的显示  不过，相差的1至2个像素在实际显示中一般不会有太大的影响，因此可以忽略不计。比较严重的错误是IE 6.0对于含有图片或者表单元等可替换行内元素的行高失效的问题，不过，在IE 7.0中已经修正了这个错误，但是其表现同其它浏览器也不相同。例如有如下代码，其显示如图7-30所示。  #lineHeight4 p { line-height : 60px; } #lineHeight4 fieldset{ border : 0; }  <div id="lineHeight4">  <p>内容含有图片在[IE 6]内浏览line-height将失效。  <img src=" ../../img/1.gif" alt="图片" width="88" height="31"/>  </p>  <form id="testForm" action="#">  <fieldset><p><label for="test1">表单元素</label>< input type=" text" maxlength="16" value="IE6内行高失效" /></p>  </fieldset>  </form> </div>  IMG_257 图7-30 包含替换元素的行高在IE内失效  由图7-30读者可以发现，IE 7.0中，将半行距分别加在了图片的上下，而由于图片默认是基线对齐，因此文字的基线下移了，这显然不符合CSS中的规定。  对于IE 6.0中行高失效的问题，需要使用CSS Hack手段来针对IE 6.0设定替换元素的上下补白来修正。  提示：关于针对IE 6的CSS Hack，请参见本书[第16章：浏览器与Hack]。 | |
| 单行文字在垂直方向居中 | 为按钮设置背景，要保证按钮中的字垂直居中显示，只需要将文字的line-height与按钮容器的高度相同即可：  <**style**>  .btn {  width: 300px;  height: 100px;**/\*\*/** background: url(**"images/btn.png"**);  background-size: 100% 100%;  text-align: center;  line-height: 100px;**/\*\*/** } </**style**> <**div** class="btn">开关</**div**> | => |
| 多行文本居中 | <**style**>  .left-tit {  width: 30px;  line-height: 158px;  text-align: center;  font-size: 14px;  color: #fff;  background: #eb5c5e;  }  .left-tit **span** {  line-height: 1;  display: inline-block;  vertical-align: middle;  font-size: 1px;  width: 14px;  } </**style**> <**div** class='left-tit'>  <**span**>多行文本垂直居中</**span**> </**div**> |  |
| 单行文本两端对齐 | 两端对齐的css属性我们知道是text-align:justify，但是这个属性有它的局限性： 1.display只能为inline或inline-block 2.多行文字才能实现两端对齐。 也就是说，在现在的情况下，小标题的单行文字仅仅使用text-align:justify是实现不了两端对齐的。 那我们就想办法把这个『单行文字』变成『多行文字』  怎样让一块内容后面多出东西，而又不会使页面多出垃圾元素呢？嘿嘿，类比清除浮动用到的方法，我们可以很快就想到使用:after伪类来解决问题。   |  |  | | --- | --- | | <**style**>  .small-unit {  padding: 10px 0;  }  .label {  position: absolute;  display: inline-block;  width: 4em;  height: 40px;  text-align: justify;  overflow: hidden;  }  .label:after {  display: inline-block;  content: **''**;  width: 4em;  height: 40px;  text-align: justify;  overflow: hidden;  }  .msg {  display: inline-block;  margin-left: 5em;  position: relative;  }  .msg:before {  content: **'：'**;  position: absolute;  left: -1em;  } </**style**> <**div** class="small-unit">  <**label** class="label">开发</**label**>  <**div** class="msg">鹅鹅鹅</**div**> </**div**> <**div** class="small-unit">  <**label** class="label">开发商</**label**>  <**div** class="msg">无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、</**div**> </**div**> <**div** class="small-unit">  <**label** class="label">开发周期</**label**>  <**div** class="msg">无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、</**div**> </**div**> | 注意点  > .label和它的after伪类要部分相同的css属性，来保证after元素是.label的第二行  {  display:inline-block;  text-align: justify;  overflow: hidden;  width: 4em;  height: 40px;  }  > :冒号推荐写在.msg元素的before伪类上，写在label里面或外面都不能满足需求，具体效果可以自己试试看  <label>{{label}}：</label>  或  <label>{{label}}</label>：  > 最后，将.label元素用绝对定位固定住，顺便实现右侧多行的效果。 | | |
| 基线对齐 | 如果行内含有图片和文字，在浏览器内浏览时，读者可以发现文字和图片在垂直方向并不是沿中线居中，而是沿基线对齐，这是因为，元素默认的垂直对齐方式为基线对齐（vertical-align: baseline）。 IMG_259 | |

line-height line-block font-size height之间的关系

|  |  |  |  |
| --- | --- | --- | --- |
| line-height、line-block | 首先看看height与line-height吧，如果对同一个div修饰的话，height指的是整个div的高度，而line-height应该指的是div中一行间距（两行文字的baseline之间的距离）。  1、line-height小于font-size：line box的优先级高，以line box为主。由于line-height是决定line box的一个很重要因素，每一行的文字将根据line-height的取值，决定最后的line box的高度。  2、line-height与垂直居中：两行baseline之间的距离包括上半行间距和下半行间距，这个就组成一个line-height。从整体上来算的话，上和下应该是一样的，试想下如果上下不相等，那该是什么样子的呢？可能可控，可以调节。但是目前没有。从美学的观点来看，今后也不会有，所以上和下半行间距是一样大小的，这就构成了垂直居中。  利用垂直居中的特性：我们可以做当行文字的居中，多行的文字居中、图片居中  3、元素对行高的影响  对于行内元素如em、strong、span和等，其padding、margin、border-top、border-bottom 不会增加行高。  padding会覆盖；margin将重置为0；border-top和border-bottom同样会覆盖。  padding-left、padding-right、border-left和border-right可用。  img元素会影响行高  设置行内元素的padding、border和margin并不会撑大外层元素的高度 | | |
| line-height、font-size | 一个空的div，<div></div>，如果没有设置至少大于1像素高度height值时，该div的高度就是个0。如果该div里面打入了一个空格或是文字，则此div就会有一个高度。  对此处高度其绝对作用的是line-height；当line-height未定义时，由font-size起作用； | | |
| line-height（px） | font-size（px） | 效果 |
| [谷歌默认大小18px] | [谷歌默认大小16px] |  |
|  | 20 |  |
| 30 | 20 |  |
| 10 | 20 |  |
| **0** | **20** |  |
|  |  |  |
| 20 |  |  |
| 20 | 0 |  |
| 20 | 10 |  |
| 20 | 40 |  |
| --谷歌浏览器：两者为空，等效于 line-height: 18px ; font-size: 16px ;  --line boxes的高度为0，但是它是以文字的水平中垂线对称分布的。这一重要的特性可以用来实现文字或图片的垂直居中对齐。 | | |
| 使用line-height代替height避免haslayout | 在某些情形下，line-height可以和height互换，因为实现的效果一样。都能撑开一个高度，然而这两个css属性有一个较隐蔽的差异，就是使用height会使标签haslayout，而使用line-height则不会。以前只有IE6的时候曾流行使用height清除浮动，就是利用了IE下height使haslayout的属性。但有时候，haslayout并不需要，反而要避免。  读过我前面有关自适应按钮文章的人可能会发现我使用了line-height代替了height，  其原因在于：IE6，IE7下，类似inline-block属性的元素里如果有block属性的元素，如果该block haslayout，则该标签会冲破外部inline-block的显示而宽度100%显示，从使按钮自适应文字大小的效果失效，解决方法就是使用line-height代替height。 height与line-height在IE6下区别  第一个标签使用height定高，结果宽度直接100%显示；  第二个标签使用line-height定高，结果很规矩，自适应与内部文字大小。  其代码如下：  <style>  .out {  display: inline-block;  background: #a0b3d6;  margin-top: 20px;  }   .in1 {  height: 20px;  }   .in2 {  line-height: 20px;  } </style> <span class="out">  <div class="in1">height:20px;</div> </span> <span class="out">  <div class="in2">line-height:20px;</div> </span> | | |
|  | 很多关于line-height的基础的知识这里并没有详细讲述，本文一开始提到的腾讯ISD团队的那篇关于行高的文章是不错的，对于了解line-height的一些特性及inline box模型很有帮助。本文更多的是讲述自己对于line-height的一些理解，简述了我使用line-height的一些经验。由于都是个人的些东西，加上本身自己的资历有限，所以可能会出现一些错误，一些遗漏之处等，还望了解。欢迎指正。要是能对您的学习有所帮助就再好不过了。 | | |

vertical-align和line-height

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | CSS深入理解vertical-align和line-height的基友关系  <http://www.zhangxinxu.com/wordpress/2015/08/css-deep-understand-vertical-align-and-line-height/> | | | | | | |
| 说明 | vertical-align支持很多属性值:  IMG_256  /\* 关键字值 \*/  vertical-align: baseline;  vertical-align: sub;  vertical-align: super;  vertical-align: text-top;  vertical-align: text-bottom;  vertical-align: middle;  vertical-align: top;  vertical-align: bottom;  /\* <长度> 值 \*/  vertical-align: 10em;  vertical-align: 4px;  /\* <百分比> 值 \*/  vertical-align: 10%;  /\* 全局值 \*/  vertical-align: inherit;  vertical-align: initial;  vertical-align: unset; | | | | | |
| 结合 CSS 2.1 规范中行内格式化上下文、行高计算、差异半差异、'line-height' 和 'vertical-align' 特性中相关说明，可以总结出行框顶边与行框底边位置的计算方式：  每一个行内元素会产生一个行内框；  行内框会在行框内横向排列；  'line-height' 特性值指定了每个行内非替换元素生成的行内框的 确切 高度；行内替换元素的高度由 'height' 特性值决定；  文字在行内框中垂直排列，上下空隙用半差异填补；如果字号大于行内框则文字从上下方向上溢出行内框，并可能渗入到其他行框内（行框是永远不会重叠的）；  'vertical-align' 特性值指定了每个行内框的垂直对齐方式；  行框的顶边界是这一系列垂直对齐的行内框最高的顶边框，底边界是最低的底边框。  行框的高度是顶边界到底边界的距离。  IMG_256  然后看看  [RD1016: 各浏览器对常用行内替换元素的 'baseline' 位置理解不同](https://link.zhihu.com/?target=http://w3help.org/zh-cn/causes/RD1016" \t "_blank)  [RD8009: IE6 IE7 IE8(Q) 中行框的顶边与底边位置识别有误](https://link.zhihu.com/?target=http://w3help.org/zh-cn/causes/RD8009" \t "_blank)  [RD3020: 在不同的文档模式中，当唯一的非表单控件类行内替换元素存在于其包容块中时，其父框的行高并不一定会计算文本基线高度](https://link.zhihu.com/?target=http://w3help.org/zh-cn/causes/RD3020" \t "_blank) | | | | | |
| vertical-align的百分比值不是相对于字体大小或者其他什么属性计算的，而是相对于line-height计算的。举个简单的例子，如下CSS代码：  {  line-height: 30px;  vertical-align: -10%;  }  实际上，等同于：  {  line-height: 30px;  vertical-align: -3px; /\* = 30px \* -10% \*/  } | | | | | |
| 1.IE6/IE7浏览器下的vertical-align的百分比值不支持小数line-height  2.vertical-align和line-height的关系是从HTML5文档声明开始的，因此，以下探讨的现象，都是在页面为HTML5声明前提下，类似下面的doctype:  <!doctype html>  <html> | | | | | |
| 未知空格 | <!doctype html> <**html**>  <**div** style="background:lightsteelblue;">  <**img** src="images/1.jpg" alt=""/> </**div**> | | |  | | |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <**html** xmlns="http://www.w3.org/1999/xhtml">  <**div** style="background:lightsteelblue;">  <**img** src="images/1.jpg" alt=""/> </**div**> | | |  | | |
| 原因 | vertical-align与line-height虽然看不见，但实际上「到处都是」；对于内联元素各种想得通或者想不通的行为表现，基本上都可以用vertical-align和line-height来解释，以及进行行为矫正。  关于上图中的未知空白具体化： | | | | | |
| <**div** style="background:lightsteelblue;">  <**img** src="images/1.jpg" alt=""/>  <**span** style="display: inline-block;background:red">Xxg</**span**> </**div**>  1.vertical-align默认值是baseline, 也就是基线对齐:字母 x的下边缘  2.字符111本身是有高度line-height的(18px)，导致图片下面就留空了 | | | |  |
| 处理 | 1.让vertical-align失效  图片默认是inline水平的，而vertical-align对块状水平的元素无感。因此，我们只要让图片display水平为block就可以了，我们可以直接设置display或者浮动、绝对定位等（如果布局允许）。例如：img { display: block; }  2.使用其他vertical-align值  告别baseline, 取用其他属性值，比方说bottom/middle/top都是可以的。  直接修改line-height值 下面的空隙高度，实际上是文字计算后的行高值和字母x下边缘的距离。因此，只要行高足够小，实际文字占据的高度的底部就会在x的上面，下面没有了高度区域支撑，自然，图片就会有容器底边贴合在一起了。比方说，我们设置行高5像素。  4. line-height为相对单位，font-size间接控制 如果line-height是相对单位，例如line-height:1.6或者line-height:160%之类，也可以使用font-size间接控制，比方说来个狠的，font-size设为大鸡蛋0, 本质上还是改变line-height值.  div { font-size: 0; } | | | | |
| **div**{  font-size: 0px; } | | **div**{  line-height: 12px;  **/\*font-size: 16px;\*//\*默认\*/** } | | |
|  | |  | | |
| **div** {  line-height: 12px;  font-size: 5px; } | | **div** {  line-height: 12px;  font-size: 0px; } | | |
|  | |  | | |
| 应用：图片垂直居中（单行文本垂直居中） | 图片的中心线为字的中心线（middle line：小写字母的x中心），离居中还存在一定偏差；  绝对居中则要将font-size设置为0，以保证content area高度是0，各种乱七八糟的线都在高度为0的这条线上，绝对中心线和中线重合。自然全垂直居中。 | | | | |
| <**style**>  .box {  line-height: 200px; }  .img {  height: 100px;  vertical-align: middle;  }  .inb {  display: inline-block;  background: red;  } </**style**> <**div** class="box">  <**img** class="img" src="images/1.jpg" alt=""/>  <**span** class="inb">Xxg0</**span**> </**div**> | <**style**>  .box {  line-height: 200px;  font-size: 0; **/\*添加其解决：中间空格 非绝对居中\*/** }  .img {  height: 100px;  vertical-align: middle;  }  .inb {  display: inline-block;  background: red;  } </**style**> <**div** class="box">  <**img** class="img" src="images/1.jpg" alt=""/>  <**span** class="inb">Xxg0</**span**> </**div**> | | | | |
|  | | | | | |
| 这种通过line-height定高，元素vertical-align:middle垂直居中的方法不仅适用于现代浏览器，连IE7浏览器也是支持的；  不过和其他浏览器再使用上还是有些需要注意的地方，就是，HTML不能这样：  <div><img src="mm1.jpg"></div>  而是需要在图片标签结束处留下空格后者换行：  <div><img src="mm1.jpg"><!-- 这里要折行或空格 -->  </div> | | | | | |
| 特别现象 | 在"[text-align:justify下列表的两端对齐布局](http://www.zhangxinxu.com/wordpress/?p=1514)"技术中，为了让任意个数的列表最后一行也是对齐排列，在列表最后会辅助列表等宽的空标签元素来占位，类似下面红色高亮HTML代码；图1  上面巨大的空隙是由占位span元素上面和下面的间隙共同组成的。  下面问题来了：上面的间隙是如何产生的？下面的间隙是如何产生的？如果去除这些间隙呢？  很多时候，复杂问题是由简单问题组合而成的，实际上，这里的间隙现象的始作俑者和上面的简单现象一样，都是vertical-align和line-height搞基带来的不好的影响。  按照之前问题解决方法，我们可以直接来个line-height:0解决垂直间隙问题：  .ul { line-height: 0; }  结果图片和图片之间的间隙是没有了，但是，图片和最后的占位元素之间依然有个几像素的间距？图2  究其原因，要说到inline-block元素和基线baseline之间的一些纠缠的关系。  设置 .justify-fix{ vertical-align:top; } 可以解决此问题；图3  具体原因见下分析； | | | | | |
| <**style**>  .ul {  width: 500px;  text-align: justify;**/\*1 两端对齐\*/** line-height: 0;**/\*2 大的间隙\*/** }  .li {  width: 128px;  }  .justify-fix {**/\*0 最后一行不对齐\*/** display: inline-block;  width: 128px;  } </**style**> <**div** class="ul">  <**img** class="li" src="images/1.jpg">  <**img** class="li" src="images/2.jpg">  <**img** class="li" src="images/3.jpg">  <**img** class="li" src="images/4.jpg">  <**img** class="li" src="images/5.jpg">  <**span** class="justify-fix"></**span**>  <**span** class="justify-fix"></**span**>  <**span** class="justify-fix"></**span**>  <**span** class="justify-fix"></**span**>  <**span** class="justify-fix"></**span**>  <**span** class="justify-fix"></**span**> </**div**> | | | | | |
| justify针对的单词与单词在一行中展示方式，连在一起的标签就如同是一个单词（<img/><img/>）；具体体现于此处img后面的换行符； | | | | | |
|  | | | | | |
| inline-block和baseline | CSS2的可视化格式模型文档中有一么一段话：  The baseline of an ‘inline-block’ is the baseline of its last line box in the normal flow, unless it has either no in-flow line boxes or if its ‘overflow’ property has a computed value other than ‘visible’, in which case the baseline is the bottom margin edge.  英文看得眼睛大，于是我中文直译了下：  ‘inline-block’的基线是正常流中最后一个line box的基线, 除非，这个line box里面既没有line boxes或者本身’overflow’属性的计算值而不是’visible’, 这种情况下基线是margin底边缘。  这段文档中出现了很多专有名词line box, line boxes等，这些是内联盒子模型中的概念，是CSS进阶必备知识。我在"[浮动深入理解(一)](http://www.zhangxinxu.com/wordpress/?p=583)"一文的中间穿插介绍了该模型。  如果大家没有足够精力去学习之，可以先看下面这张图：  默认图文line boxes示意  由于上面的译文是直译的，理解起来还是有些拗口，我使用通俗的话描述就是：一个inline-block元素，如果里面没有inline内联元素，或者overflow不是visible，则该元素的基线就是其margin底边缘，否则，其基线就是元素里面最后一行内联元素的基线。  纳尼，还是没反应过来？  那我们看下面这个例子，应该就知道什么意思了。  两个同尺寸的inline-block水平元素，唯一区别就是一个空的，一个里面有字符，代码如下：  .dib-baseline {  display: inline-block; width: 150px; height: 150px;  border: 1px solid #cad5eb; background-color: #f0f3f9;  }  <span class="dib-baseline"></span>  <span class="dib-baseline">x-baseline</span>  结果，科科：    会发现，明明尺寸、display水平都是一样的，结果呢，两个却不在一个水平线上对齐，为什么呢？哈哈，上面的规范已经说明了一切。第一个框框里面没有内联元素，因此，基线就是容器的margin下边缘，也就是下边框下面的位置；而第二个框框里面有字符，纯正的内联元素，因此，第二个框框就是这些字符的基线，也就是字母x的下边缘了。于是，我们就看到了框框1下边缘和框框2里面字符x底边对齐的好戏。框框2有个小彩蛋，点击可以toggle其innerHTML，会发现，如果框框2里面没文字，就和框框1举案齐眉了。    下面我们要做一件很有必要的事情，用来帮助我们理解上面复杂例子在line-height值为0后的表现，什么事情呢？哈，同境界模拟，我们也设置框框2的line-height值为0，于是，就会是下面这样的表现：    知道框框2为何又下沉了一点吗？  因为字符实际占据的高度是由行高决定的，当行高变成0的时候，字符占据的高度也是0，此时，高度的起始位置就变成了字符content area的垂直中心位置，于是，文字就一半落在看看2的外面了。 IMG_257  由于文字字符上移了，自然基线位置(字母x的底边缘)也往上移动了，于是，两个框框的垂直落差就更大了。  OK，明白了上面的简单例子，也就能明白上面的复杂例子。紧接着，如果我们在最后一个占位的<i>元素后面新增同样的x-baseline字符，则：    大家是不是就可以明白原因所在啦！  额~居然还有小伙伴皱眉头，那我再用文字解释下：  现在行高line-height是0, 则最后的x-baseline的垂直中线就和上面一列的图片对齐，而基线呢，就在中线下面差不多半个x的高度地方，而这个高度落差就是最后图片和容器的间隙高度值，因为前面的<i class="justify-fix">是个空元素，基线是自身的底部，哈哈，造业啊！  IMG_262  OK，一旦知道了现象的本质，我们就能轻松对症下药了！要么改造占位<i>元素的基线、要么改造"幽灵空白节点"的基线位置、要么使用其他vertical-align对齐方式~  首先，来个最有意思的方法，对吧，改造占位<i>元素的基线。这个很简单，对吧，只要在空的<i>元素里面随便放几个字符就可以了，例如，里面有个x：    会发现，间隙没有了！IMG_267 为什么呢？哈哈，因为<i>元素的基线和"幽灵空白节点"的基线位置现在一致了，没有了错位，自然就不会有间隙啦！  改造"幽灵空白节点"的基线位置，哈哈，使用font-size，字体足够小时，基线和中线会重合在一起，什么时候字体足够小呢，就是0. 于是，CSS代码(line-height如果是相对值，line-height:0也可以省掉)：  div { font-size: 0; }  IMG_268 IMG_269 IMG_270IMG_271  使用其他vertical-align对齐方式，就是让两端对齐的列表元素vertical-align:top/bottom/...之类。  div { line-height: 0; }  .justify-fix { display: inline-block; width: 128px; vertical-align: top; }  最后的效果是：  IMG_272 IMG_273 IMG_274IMG_275  恩恩，各种方法都完美解决了垂直间隙的问题，来，各个大大的赞！ | | | | | |
|  |  | | | | | |

vertical-align

|  |  |  |
| --- | --- | --- |
|  | <https://segmentfault.com/a/1190000002668492>  [[翻译]关于Vertical-Align你需要知道的事情](https://segmentfault.com/a/1190000002668492)  效果：<https://jsfiddle.net/medifle/erpo7u4h/1/> | |
| 问题来了 | **在遇到将元素在垂直方向上对齐的需求时，CSS提供了多种方法，有时候我用float解决，有时候我使用position:absolute解决，还有时候采用的方法是手动调整margin和padding。**  其实我并不喜欢这些解决方法，float会让元素的顶端对齐而且需要手动清楚浮动。绝对定位让元素脱离了正常流，这样这些元素就不会再影响到周围的元素。在元素的padding和margin固定的情况下，一点小的改变也很有可能对布局造成影响。  Vertical-align通常来说使用这个属性进行布局是一种hack行为，因为它本来并不是被用于这个目的。它用在文本和与文本相邻元素的垂直方向上的对齐问题。然而，你也可以使用Vertical-align在不同的上下文中对元素进行灵活的，细粒度的排布。元素的尺寸无需知晓，元素任然处于正常流当中，因此元素的变化会影响到周围的元素，这使得Vertical-align是非常有价值的排布方案的选择。 | |
|  | Vertical-align被用于垂直对齐inline元素，也就是display值为inline和inline-block的元素。inline-table的元素不在本文的讨论范围内。  inline-block元素就像它的名字一样，同时具备inline元素和block元素的特点，这样的元素有padding,margin,border,width,height。其中高度有可能是由元素的内容决定的。  inline元素基本上值得就是文本,inline元素一个挨着一个的摆放在行内，当行内元素太多的情况下，一个新行会被创建出来，这些行也叫做line--box。它将行内的所有内容都包裹了起来。根据行内内容的不同，line-box的尺寸也会不同，在接下来的图里面，红线代表了line-box的上下边界在这些line-box内，Vertical-align属性负责对齐一些独立的元素。    那么，这些元素是要和谁对齐呢？ | |
| 边界和baseline | 对于垂直对齐这个知识点来说最重要的就是涉及元素的baseline。有时候元素的盒模型的上下边界也会变的很重要。    如图所示有三行文字。行高的上下边界是红线。文字的上下边界是绿色的线，蓝色的线就是baseline了，  > 左边文字的高度与行高是一致的，因此绿线和红线重合了，  > 中间的行高是文字大小的两倍，  > 右边，行高是文字大小的二分之一。  行内元素的外边缘在行高的上边缘和下边缘这个范围内对齐，如果行高小于文字的高度也无所谓。  关于baseline的定义还是直接给出[标准的链接](http://www.w3.org/TR/CSS2/visudet.html" \l "leading" \t "https://segmentfault.com/a/_blank):https://www.w3.org/TR/CSS2/visudet.html#leading | |
| inline-block元素 | 从左到右的三幅图片都是inline-block元素，不同的是，  > 左面包含着没有脱离正常流的内容c，  > 中间的除了没有脱离正常流的内容以外还加了overflow:hidden,  > 右面的没有内容但是内容区还有高度。  红线代表了margin-box的边界。黄色代表的是border，绿色的是padding，蓝色的是content，蓝色的线代表的还是baseline。  inline-block元素的外边缘就是margin-box的边缘。  inline-block元素的baseline的位置要看该元素有没有处于正常流之内的内容。  (1)在有处于正常流内容的情况下，inline-block元素的baseline就是最后一个作为内容存在的元素的baseline，这个元素的baseline的确定就要根据他自身来确定了。  (2)在overflow属性不为visible的情况下，baseline就是margin-box的下边界了。  (3)第三种情况下baseline还是margin-box的下边界。 | |
| line-box | 关于line-box的图上面已经给过了，这次我将文字部分高亮显示。line-box的上边界与最高元素的上边界对齐，下边界与最低元素的下边界对齐。  W3C标准中并没有定义line-box的baseline的位置。这一点很让人困惑，baseline的位置需要满足vertical-align属性的值以及让line-box的高度最小等条件，是一个很灵活的参数。  line-box的baseline是不可见的，但是可以很轻松的将它可视化出来，在行的开头添加一个字母，比如'x'，这个字母的下边界默认就是baseline的位置。围绕着baseline在line-box中形成了文字盒。文字盒可以被认为是没有和任何元素对齐的line-box中的inline元素，它的告诉与它的父元素的font-size的值相同。因此，文字盒仅仅包含非格式化的line-box的文本，文字盒的边界由绿线来表示。因为文字盒是紧挨着baseline的，所以baseline的位置发生变化的话，文字盒的位置也会跟着改变(这里所说的文字盒在标准中被叫做[strut](http://www.w3.org/TR/CSS21/visuren.html" \l "positioning-scheme" \t "https://segmentfault.com/a/_blank))。 总结起来的话有以下两点：  有一个区域叫做line-box，垂直方向上的对齐都是发生在这个区域里面，它有baseline，有文字盒，有上下边界。  inline元素也有baseline和上下边界，inline元素是需要对齐的对象。 | |
|  | <**style**>  .v {  display: inline-block;  min-width: 20px;  background: lightgrey;  }  .v1 {  vertical-align: baseline;  }  .v2 {  height: 50px;  vertical-align: text-top;   }  .v3 {  vertical-align: top;   }  .v4 {  height: 60px;  vertical-align: text-bottom;   }  .v5 {  vertical-align: bottom;   } </**style**> <**span** class="v v1">this</**span**> <**span** class="v v2"></**span**> <**span** class="v v3">can</**span**> <**span** class="v v4"></**span**> <**span** class="v v5">happen</**span**> | 可以将height换成font-size line-height进行测试； |
| Vertical-align属性的值 | 有关Vertical-align各个取值的说明读者还是上MDN看吧。也可以看看天镶大神的[博客上的文章](http://lingyu.wang/2014/04/13/vertical-align/" \t "https://segmentfault.com/a/_blank) | |
| Vertical这个属性到底是怎么起作用 | 对齐一个Icon  我想将一个icon与文字对齐仅仅使icon的vertical-align属性的值设置为middle似乎不能产生令人满意的结果，例子如下图       |  | | --- | | <**style**>  .icon {  display: inline-block;  height: 30px;  width: 30px;  }  .icon-middle {  vertical-align: middle;  }  .text-middle{  vertical-align: middle;  } </**style**>   **<!--icon中有文字，效果将不同-->** <**span** class="icon icon-middle"></**span**><**span**>text</**span**> <**br**/> <**br**/> <**span** class="icon icon-middle"></**span**><**span** class="text-middle">text</**span**> | | 图片icon的高度：10px - 16px - 30px |   原因就是左面的文字根本就没有发生对齐行为，它还是对齐于line-box的baseline。而vertical-align对齐的点是baseline加上半个x的距离(half of the x-height)。因此文字的最高点超过了icon的高度。  而右面的例子，文字与icon都对齐于一个中点，文字的baseline稍微下移，位于line-box的baseline的下方。结果是很好的达到了icon与文字对齐的效果。  这是一个Vertical-align的坑，line-box中的所有元素都会影响到baseline的位置。假设，一个元素按某种方式垂直对齐了，但是这种对齐方式会引起baseline的移动，又因为大部分的垂直对齐方式(除了top和bottom)和baseline有关，因此这个元素的垂直方向对齐的行为会引起该line-box内其他元素位置的调整。 下面还是一些例子  一个很高的元素，其高度占满了整个line-box，那么vertical-align对其实没有影响的，在它的top和bottom之外没有空间让其移动。但是为了满足它的vertical-align的值，line-box的baseline会发生移动，左面的高元素的取值为text-bottom，矮元素的取值为baseline。右面的高元素的取值为text-top，你会看到baseline跳上去了    <!-- left mark-up -->  <span class="tall-box text-bottom"></span>  <span class="short-box"></span>  <!-- right mark-up -->  <span class="tall-box text-top"></span>  <span class="short-box"></span>  <style type="text/css">  .tall-box,  .short-box { display: inline-block;  /\* size, color, etc. \*/ }  .text-bottom { vertical-align: text-bottom; }  .text-top { vertical-align: text-top; }  </style>  如果把高元素的vertical-align设置为其他值，也能看到类似的行为 甚至将vertical-align设置为bottom或者是top也会让baseline发生移动。这很奇怪，因为这时候应该就没baseline什么事儿了。    <!-- left mark-up -->  <span class="tall-box bottom"></span>  <span class="short-box"></span>  <!-- right mark-up -->  <span class="tall-box top"></span>  <span class="short-box"></span>  <style type="text/css">  .tall-box,  .short-box { display: inline-block;  /\* size, color, etc. \*/ }  .bottom { vertical-align: bottom; }  .top { vertical-align: top; }  </style> | |
|  | 将两个更大的元素放在一个line里面，并且设置vertical-align的值让line-box的baseline移动。在满足vertical-align数值对齐的条件下，line-box的高度会自我调整，如左面的图。再增加第三个元素，第三个元素如果因为其vertical-align的设置不会超过line-box的边缘的话，它是不会影响到line-box的高度和baseline的位置的，如果它会超过line-box的边缘，那么line-box的高度和baseline的位置也会进行调整。在第二种情况下，另外两个元素的位置发生了下移。    <**style** type="text/css">  .tall-box {  display: inline-block;  width: 20px;  height: 50px;  **/\* size, color, etc. \*/** }  .middle {  vertical-align: middle;  }  .text-top {  vertical-align: text-top;  }  .text-bottom {  vertical-align: text-bottom;  }  .text-100up {  vertical-align: 100%;  } </**style**>  **<!-- left mark-up -->** <**span** class="tall-box text-bottom"></**span**> <**span** class="tall-box text-top"></**span**>  **<!-- mark-up in the middle -->** <**span** class="tall-box text-bottom"></**span**> <**span** class="tall-box text-top"></**span**> <**span** class="tall-box middle"></**span**>  **<!-- right mark-up -->** <**span** class="tall-box text-bottom"></**span**> <**span** class="tall-box text-top"></**span**> <**span** class="tall-box text-100up"></**span**> | |
| 下空隙 | inline元素下方可能会有一点空隙  下面给出一个例子，如果尝试将li元素在垂直方向上进行对齐的话在，这个现象非常常见。  正如你所见，li元素是对齐baseline的，baseline的下方会给字母的一部分留出空间，因此会产生一个空隙。解决方案就是改变line-box的baseline的位置，比如将这些li这是为vertical-align:middle  inline元素之间的空隙造成布局效果与理想状态发生偏差  这主要是因为inline元素本身的原因，但是因为如果要让vertical-align的值产生作用的话，inline元素是必备要求，因此了解一点也不错嘛  两个建议，如果vertical-align属性不起作用，那么问问自己：  line-box的上下边缘以及baseline的位置在哪里？  inline元素的上下边缘以及baseline的位置在哪里？   |  |  | | --- | --- | |  | | | <**style** type="text/css">  .ul{  list-style: none;  padding: 0;  }  .li {  display: inline-block;  height: 30px;  width: 10px;  **/\* size, color, etc. \*/** } </**style**> <**ul** class="ul">  <**li** class="li"></**li**>  <**li** class="li"></**li**>  <**li** class="li"></**li**> </**ul**> |  | | <**style** type="text/css">  .ul{  list-style: none;  padding: 0;  }  .li {  display: inline-block;  height: 30px;  width: 10px;  vertical-align: top;**/\*middle:若元素的高度比默认的字体的高度还小，则会导致上方空出一块\*/  /\* size, color, etc. \*/** } </**style**> <**ul** class="ul">  <**li** class="li"></**li**>  <**li** class="li"></**li**>  <**li** class="li"></**li**> </**ul**> |  | | <**style** type="text/css">  .ul{  list-style: none;  padding: 0;  }  .li {  display: inline-block;  height: 10px;  width: 10px;  vertical-align: middle;**/\*middle:若元素的高度比默认的字体的高度还小，则会导致上方空出一块\*/  /\* size, color, etc. \*/** } </**style**> <**ul** class="ul">  <**li** class="li"></**li**>  <**li** class="li"></**li**>  <**li** class="li"></**li**> </**ul**> |  | | |
| 中间空隙 | 这个空隙来源于inline元素之间的空格，所有的inline元素之间的空白都会变成一个空格。如果你想让inline元素在水平上紧挨着，设置它们的宽度是不行的。因为之间存在空隙，所以行的宽度不够放下两个inline元素。一行会被破坏为两行。解决方案如下图的右侧    <!-- left mark-up -->  <div class="half">50% wide</div>  <div class="half">50% wide... and in next line</div>  <!-- right mark-up -->  <div class="half">50% wide</div><!----><div class="half">50% wide</div>  <style type="text/css">  .half { display: inline-block;  width: 50%; }  </style> | |
|  |  | |

## 关于table布局

table布局

|  |  |  |
| --- | --- | --- |
| table布局与css布局 | Tables的缺点  1、Table要比其它html标记占更多的字节。(延迟下载时间，占用服务器更多的流量资源。)  2、Tablle会阻挡浏览器渲染引擎的渲染顺序。(会延迟页面的生成速度，让用户等待更久的时间。)  3、Table里显示图片时需要你把单个、有逻辑性的图片切成多个图。(增加设计的复杂度，增加页面加载时间，增加HTTP会话数。)  4、在某些浏览器中Table里的文字的拷贝会出现问题。(这会让用户不悦。)  5、Table会影响其内部的某些布局属性的生效(比如<td>里的元素的height:100%)(这会限制你页面设计的自由性。)  6、一旦学了CSS知识，你会发现使用table做页面布局会变得更麻烦。(先花时间学一些CSS知识，会省去你以后大量的时间。)  7、table对对于页面布局来说，从语义上看是不正确的。(它描述的是表现，而不是内容。)  8、table代码会让阅读者抓狂。(不但无法利用CSS，而且会你不知所云)  9、table一旦设计完成就变成死的，很难通过CSS让它展现新的面貌。 | |
| 由于html文件中的table标签的浏览速度较慢，所以，使用嵌套表格的方法来布局网页框架会使网页浏览的速度变慢。因为table是中的内容是自适应的，为了自适应，它要计算嵌套最深的节点以满足自适应，所以有可能会有一断时间出现空白才显示。所以是会有问题的。  使用DIV的方法布局网页框架的优点：可以通过css样式给框架进行功能强大的属性设置以及给网页的局部进行任意的定位，制作出来的页面浏览速度较快，同时页面的风格可以通过修改单独的css文件进行随意的修改和更新；缺点：每个div容器都需要定义css样式来控制，制作过程相比table方式要复杂。现在的网页越来越倾向于使用DIV的方法来布局网页了。  论代码的复杂程度来说，不管table还是div都是很冗长的。css的网页比table的网页体积也许要小一点，但是复杂程度相差不大。网络访问上css网页稍微会快点，但是总体的用户体验感觉差别不是很大。搜索引擎优化方面，我曾经用过cms数据采集，里面的程序用的HtmlParser类库解析html网页，然后将元素中的数据用[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/aoxida/article/details/_blank)dom转化为xml导入到[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/aoxida/article/details/_blank)。这个类似蜘蛛的程序在爬行的时候table数据和div中数据爬行起来效率是一样的，现在的大型搜索引擎不会因为div这种趋势而抛弃table我认为，如果一个网页[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/aoxida/article/details/_blank)的不好，不论table还是div都会被引擎所抛弃。 　 但是网上的大部分人对于div+css的开发时间上是table的很多倍，而且还一个问题，网友说了，需要写很多代码用于适应不同浏览器，这个我现在维护的公司网站在早期也有这个问题。其实这个问题是是否熟悉div的问题，如果一个熟悉div的话应该不会比table慢太多。 　 "W3C规范中说Table是用来传数据的，但并没有指出Table不可以用来布局。搜索引擎对于网页的收录和排序显然不是以是否采用表格和CSS定位来衡量，这就是为什么很多传统表格布局制作的网站在搜索结果中的排序靠前，而很多使用CSS及web标准制作的网页排名依然靠后的原因。因为对于搜索引擎而言，网站结构、内容、相关网站链接等因素始终是网站优化最重要的指标。  汇总：  table优点：开发时间短（使用DW开发速度快）；纯table各浏览器不会有兼容问题；内容可自适应；在搜索引擎排名能靠前  table缺点：如果布局变更，需要重新开发；如果table里有div ul 等，可能会出现浏览器兼容问题；加载速度慢；table套table，会害死维护人员的  div优点：内容和显示分离，便于维护扩展，网页布局方便，当需求改变的时候，效果最明显 | |
| table-layout:fixed; | 设置表格布局算法： table { table-layout:fixed; }  定义和用法  tableLayout 属性用来显示表格单元格、行、列的算法规则。  固定表格布局：  固定表格布局与自动表格布局相比，允许浏览器更快地对表格进行布局。  在固定表格布局中，水平布局仅取决于表格宽度、列宽度、表格边框宽度、单元格间距，而与单元格的内容无关。  通过使用固定表格布局，用户代理在接收到第一行后就可以显示表格。  自动表格布局：  在自动表格布局中，列的宽度是由列单元格中没有折行的最宽的内容设定的。  此算法有时会较慢，这是由于它需要在确定最终的布局之前访问表格中所有的内容。  说明  该属性指定了完成表布局时所用的布局算法。固定布局算法比较快，但是不太灵活，而自动算法比较慢，不过更能反映传统的 HTML 表。   |  |  | | --- | --- | | 可能的值： | | | automatic | 默认。列宽度由单元格内容设定。 | | fixed | 列宽由表格宽度和列宽度设定。 | | inherit | 规定应该从父元素继承 table-layout 属性的值。 | | |
| <**style** type="text/css">  **table**.one {  table-layout: automatic;  }  **table**.two {  table-layout: fixed;  } </**style**> <**table** class="one" border="1" width="100%">  <**tr**>  <**td** width="20%">100000000000000000</**td**>  <**td** width="40%">10000000</**td**>  <**td** width="40%">100</**td**>  </**tr**> </**table**> <**table** class="two" border="1" width="100%">  <**tr**>  <**td** width="20%">100000000000000</**td**>  <**td** width="40%">10000000</**td**>  <**td** width="40%">100</**td**>  </**tr**> </**table**> |  |
| <col> 标签 | col 元素为表格中的三个列规定了不同的对齐方式：  <**table** id="myTable" border="1" width="100%">  <**col** width="10%" align="center">  <**col** width="40%">  <**col** width="40%">  <**tr**>  <**td**>222</**td**>  <**td**>10000000</**td**>  <**td**>100</**td**>  </**tr**> </**table**>  浏览器支持  所有主流浏览器都支持 <col> 标签。  定义和用法  <col> 标签为表格中一个或多个列定义属性值。  如需对全部列应用样式，<col> 标签很有用，这样就不需要对各个单元和各行重复应用样式了。  您只能在 table 或 colgroup 元素中使用 <col> 标签。  提示和注释：  提示：请为 <col> 标签添加 class 属性。这样就可以使用 CSS 来负责对齐方式、宽度和颜色等等。  提示：如果您希望在 colgroup 内部为每个列规定不同的属性值时，请使用此元素。如果没有 col 元素，列会从 colgroup 那里继承所有的属性值。  注释：col 元素是仅包含属性的空元素。如需创建列，您就必须在 tr 元素内部规定 td 元素。  可选的属性   |  |  |  | | --- | --- | --- | | [align](http://www.w3school.com.cn/tags/att_col_align.asp" \o "HTML <col> 标签的 align 属性) | right  left  center  justify  char | 规定与 col 元素相关的内容的水平对齐方式。 | | [valign](http://www.w3school.com.cn/tags/att_col_valign.asp" \o "HTML <col> 标签的 valign 属性) | top  middle  bottom  baseline | 定义与 col 元素相关的内容的垂直对齐方式。 | | [width](http://www.w3school.com.cn/tags/att_col_width.asp" \o "HTML <col> 标签的 width 属性) | pixels  %  relative\_length | 规定 col 元素的宽度。 | |  |  |  | | [char](http://www.w3school.com.cn/tags/att_col_char.asp" \o "HTML <col> 标签的 char 属性) | character | 规定根据哪个字符来对齐与 col 元素相关的内容。 | | [charoff](http://www.w3school.com.cn/tags/att_col_charoff.asp" \o "HTML <col> 标签的 charoff 属性) | number | 规定第一个对齐字符的偏移量。 | | [span](http://www.w3school.com.cn/tags/att_col_span.asp" \o "HTML <col> 标签的 span 属性) | number | 规定 col 元素应该横跨的列数。 |   经测试，align无效？ | |
|  | 多级表头  table元素是通过colSpan和rolSpan控制每个单元格左占列和行的多少。  <table>  <thead>  <tr>  <th colspan="1" rowspan="2"></th>  <th colspan="2" rowspan="1"></th>  </tr>  <tr>  <th colspan="1" rowspan="1"></th>  <th colspan="1" rowspan="1"></th>  </tr>  </thead>  </table>  [IMG_256](http://og40ypzfa.bkt.clouddn.com/table.png)  要实现多级表头，则同样需要利用这种特性，它将依赖一个二维数组。因此需要将columns这个树结构根据树的层级建立一个二维数组。  [IMG_257](http://og40ypzfa.bkt.clouddn.com/table.2.png) | |

基于display:table的CSS布局

|  |  |  |
| --- | --- | --- |
|  | 2012 11-28基于display:table的CSS布局  <http://www.frontopen.com/331.html> | |
| 概念 | 当IE8发布时，它将支持很多新的CSS display属性值，包括与表格相关的属性值：table、table-row和table-cell，它也是最后一款支持这些属性值的主流浏览器。它标志着复杂CSS布局技术的结束，同时也给了HTML表格布局致命一击。最终，使用CSS布局来制作出类似于table布局的栅格将会变得十分迅速和简单。  网页元素应用上那些与表格相关的display属性值后，能够模仿出与表格相同的特性。我将会在该文中给大家演示这种方法给CSS布局带来的巨大影响。  使用CSS表格  CSS表格能够解决所有那些我们在使用绝对定位和浮动定位进行多列布局时所遇到的问题。例如，"display:table;"的CSS声明能够让一个HTML元素和它的子节点像table元素一样。使用基于表格的CSS布局，使我们能够轻松定义一个单元格的边界、背景等样式，而不会产生因为使用了table那样的制表标签所导致的语义化问题。 | |
| demo | <**style**>  #main {  display: table;  width: 100%;  border-collapse: collapse;  }  #nav {  display: table-cell;  width: 25%;  background-color: #e7dbcd;  }  #extras {  display: table-cell;  width: 25%;  padding-left: 10px;  border-right: 1px dotted #d7ad7b;  }  #content {  display: table-cell;  width: 50%;  padding-left: 10px;  } </**style**> <**div** id='main'>  <**div** id='nav'>1 <**img** src="images/1.jpg" alt=""/></**div**>  <**div** id='extras'>2 <**img** src="images/2.jpg" alt=""/></**div**>  <**div** id='content'>3 <**img** src="images/3.jpg" alt=""/></**div**> </**div**> | |
| > 这种基于表格的新CSS布局方式能够正确的在IE8、Firefox、Safari和Opera（译者注：包括FF2/FF3/Google都通过了测试）中显示出来。  > 轻松实现了三栏等高布局，而无需使用伪造背景图片之类的技巧，更不用担心定位和清除浮动的问题 | |
| 相关属性 | 有用的表格属性  当使用CSS表格时，因为这些元素遵从table布局的普通规则，所以你还可以给它们应用其它表格相关的CSS属性。下面是一些派得上用场的属性：  table-layout  将table-layout属性设置为fixed可以让浏览器按照固定算法来渲染单元格的宽度。这在固定宽度布局中非常有用，例如我们最上面的那段布局代码。  border-collapse  和普通的HTML表格一样，你可以使用border-collapse属性来定义你的table布局元素之间使用何种形式的边框，是共用边框（赋值为collapse）还是使用各自独立的边框（赋值为separate）。  border-spacing  如果你声明了"border-collapse:separate;"，那么你就可以使用border-spacing属性来定义相邻两个单元格边框间的距离。 | |
|  | 可用的display属性值：  table使该元素按table样式渲染  table-row使该元素按tr样式渲染  table-cell使该元素按td样式渲染  table-row-group使该元素按tbody样式渲染  table-header-group使该元素按thead样式渲染  table-footer-group使该元素按tfoot样式渲染  table-caption使该元素按caption样式渲染  table-column使该元素按col样式渲染  table-column-group使该元素按colgroup样式渲染 | |
|  | 可能你会对我们上面给出的布局实例有点不爽——毕竟，正如我自己也是一名WEB标准化的拥护者，我们不都一直坚持不应该使用table来进行布局吗？  table元素在HTML当中是一个包含语义的标签：它描述什么是数据。因此，你只能用它来标记那些需要制表的数据，例如一张财务信息表。如果数据能够以电子表格的形式保存在你的电脑中，那它在HTML文档中很可能需要用到table标签进行标记。  从另一方面来看，display的table属性值只是声明了某些元素在浏览器中的样式——它不包含语义。如果使用table元素来进行布局，它将会告诉客户端：这些数据是制表的。使用一些display属性被设置为table和table-cell之类的div标签，除了告诉客户端以某种特定的样式来渲染它们以外，不会告诉客户端任何语义，只要客户端能够支持这些属性值。  当然，我们同样还要注意，当我们真的需要制表数据的时候不要使用一大堆被声明为display:table;的div元素。  我们上面的那个例子是一个简单的单行三列布局，无需费尽心思，我们就能够使用这种技术轻松实现复杂的栅格布局。 | |
| 匿名表格元素 | 匿名表格元素  CSS表格除了包含table布局的普通规则之外，同时还有着CSS table布局的超强特性：缺少的表格元素会被浏览器以匿名方式创建。CSS2.1规范中写道：  CSS2.1表格模型中的元素，可能不会全部包含在除HTML之外的文档语言中。这时，那些"丢失"的元素会被模拟出来，从而使得表格模型能够正常工作。所有的表格元素将会自动在自身周围生成所需的匿名table对象，使其符合table/inline-table、table-row、table-cell的三层嵌套关系。  这段话的意思是，如果我们为元素使用"display:table-cell;"属性，而不将其父容器设置为"display:table-row;"属性，浏览器会默认创建出一个表格行，就好像文档中真的存在一个被声明的表格行一样。 | |
| 创建匿名表格元素的规则  这些匿名的盒对象不是用魔术变出来的，它们也不会自动往你的HTML源码中添加新的标签。为了完全发挥出匿名表格元素的优势，你最好能够对创建它们的规则有所了解。如果布局中调用了匿名元素，浏览器将会根据需要创建一个匿名的盒对象并将它的CSS display属性设置为table、table-row或table-cell中的一个。  如果某个元素已经被设置为"display:table-cell;"，而它的父节点（包含它的容器）没有被设置为"display:table-row;"属性，那么浏览器将会创建一个被设置为"display:table-row;"的匿名盒对象来嵌套它。并且与之相邻的属性为"display: table-cell;"的兄弟节点也都会被这个匿名盒对象所包含，直到碰到一个没有被设置为"display: table-cell;"的元素而结束这一行。以下是相关的代码样例：  <div class="cell">CELL A</div>  <div class="cell">CELL B</div>  <div class="cell">CELL C</div>  <div>Not a cell</div>  上面的三个类名为"cell"的div元素均被设置为"display:table-cell;"，它们将会像一个单行表格的三个单元格一样并列排布。最后一个div元素则不会被包含在这一表格行当中，因为它没有被设置成"display:table-cell;"。  如果某个元素被设置为"display:table-row;"，而它的父节点没有被设置为"display:table;"（或者"display:table-row-group;"），浏览器将会创建一个被设置为"display:table;"的匿名盒对象来嵌套它，与之相邻的属性为"display: table-row;"的兄弟节点也都会被包含其中。同样，如果某个元素被设置为"display:table-row;"，但它的内部却缺少"display:table-cell;"的元素，那么一个匿名的table-cell将会被创建，用来包含该table-row中的所有元素。  请看以下代码：  <div class="row">ROW A</div>  <div class="row">ROW B</div>  <div>Not a row</div>  上面两排类名为"row"的div元素被设置了"display:table-row;"属性，它们将会像单列表格中的两行一样依次排列。最后一个div元素则不会包含在这个匿名的table中。  以此类推，如果某个元素的display属性值被设置为与表格相关的值，如table-row-group、table-header-group、 table-footer-group、table-column、table-column-group以及table-caption，但同时又没有一个被设置为"display:table;"的父元素，那么一个匿名的盒对象将会被创建用来包含该元素和它的某些兄弟节点。 | |
| 栅格 | <**style**>  .grid {  display: table;  border-collapse: separate;  border-spacing: 60px 20px; **/\*设置间距\*/** }  .row {  display: table-row;  }  .li {  display: table-cell;  width: 240px;  background-color: lightgrey;  vertical-align: top;  text-align: center;  }  .li **p** {  color: #fff;  font-size: 85%;  text-align: left;  padding-top: 8px;  } </**style**> <**div** class='grid'>  <**div** class='row'>  <**div** class='li'>  <**img** src='images/1.jpg' alt='A Lily'/>  <**br**/>  A lily in the gardens of The Vyne Country House  </**div**>  <**div** class='li'>  <**img** src='images/2.jpg' alt='A Fuchsia plant'/>  <**br**/>  Fuchsia plant in my garden  </**div**>  </**div**>  <**div** class='row'>  <**div** class='li'>  <**img** src='images/2.jpg' alt='A crazy looking Allium flower'/>  <**br**/>  A crazy looking flower  </**div**>  <**div** class='li'>  <**img** src='images/3.jpg' alt='A Robin sitting on a fence'/>  <**br**/>  This robin has been visiting our garden over the summer.  He is very friendly and doesn’t seem to be too worried about sharing the garden with us.  </**div**>  </**div**> </**div**> | 制作完美的栅格  制作等高栅格对于传统CSS布局技术来说已经成为一个难题，然而使用合适的CSS表格则很容易实现。例如，如果我们想制作一个包含图片和标题的影像图库栅格（如下图），使用CSS表格很快就能搞定。  每张图片被一个img标签引用，它的标题包含在P元素中，它们均被包含在一个类名为“li”的div元素中。同一行的div被一个类名为“row”的div元素包含，整个影像图库被一个类名为“grid”的div元素包含。  CSS代码简明易懂，可能你还注意到了我们是怎样通过border-spacing属性来控制单元格图像之间的距离的。制作一张栅格布局变得再简单不过了，同时我们还可以避免那些使用float元素实现等高布局所带来的麻烦。  效果 |

table-cell

|  |  |  |  |
| --- | --- | --- | --- |
|  | 我所知道的几种display:table-cell的应用  <http://www.zhangxinxu.com/wordpress/2010/10/%E6%88%91%E6%89%80%E7%9F%A5%E9%81%93%E7%9A%84%E5%87%A0%E7%A7%8Ddisplaytable-cell%E7%9A%84%E5%BA%94%E7%94%A8/>  一、display:table-cell属性简述  二、display:table-cell与大小不固定元素的垂直居中  二、display:table-cell与两栏自适应布局  三、display:table-cell下的等高布局  四、display:table-cell下的列表布局 | | |
|  | display:table-cell属性指让标签元素以表格单元格的形式呈现，类似于td标签。目前IE8+以及其他现代浏览器都是支持此属性的，但是IE6/7只能对你说sorry了，这一事实也是大大制约了display:table-cell属性在实际项目中的应用（写于2016年8月24日：时代在变，当下，是时候登上舞台了）。  其特殊之处就在于垂直居中，等高，水平级联，匿名创建等特性；  我们都知道，单元格有一些比较特别的属性，例如元素的垂直居中对齐，关联伸缩等，所以display:table-cell还是有不少潜在的使用价值的，虽说IE6/7不支持此属性，但是幸运的是，IE6/7一些乱糟糟的属性与渲染，我们可以其他方法实现同样或是类似的效果。  与其他一些display属性类似，table-cell同样会被其他一些CSS属性破坏，例如float, position:absolute，所以，在使用display:table-cell与float:left或是position:absolute属性尽量不用同用。设置了display:table-cell的元素对宽度高度敏感，对margin值无反应，响应padding属性，基本上就是活脱脱的一个td标签元素了。 | | |
| display:table-cell与大小不固定元素的垂直居中 | 使用display:table-cell让大小不固定元素垂直居中已经是很老的方法了，关于此应用，我已经在"[大小不固定的图片、多行文字的水平垂直居中](http://www.zhangxinxu.com/wordpress/?p=61)"这篇文章中有过介绍。 | | |
| <**style**>  .box {  display: table-cell;  width: 300px;  height: 300px;  text-align: center;  vertical-align: middle;**/\*内部元素居中显示\*/** }  .box .img {  height: 200px;  vertical-align: middle;**/\*行内元素居中\*/** } </**style**>   <**div** class="box">  <**img** class="img" src="images/1.jpg" alt=""/>x </**div**> | |  |
| <**style**>  .is-Table {  display: table;  width: 300px;  height: 300px;  }  .is-Table .Table-Cell {  display: table-cell;  text-align: center;**/\*table-cell内部元素（行级）居中\*/** vertical-align: middle;**/\*垂直居中\*/** }  .is-Table .Center-Block {  display: inline-block;  width: 80%;  } </**style**> <**div** class="is-Table">  <**div** class="Table-Cell">  <**div** class="Center-Block">  <**img** src="images/1.jpg" alt=""/> a s d f g h j k l  </**div**>  </**div**> </**div**> | |  |
| display:table-cell与两栏自适应布局 | 就在前不久，看facebook好友动态列表页面前端代码的时候才发现原来display:table-cell可以用在两栏的自适应布局上。  虽然IE6/7不认识display:table-cell，但是亏了其一向自以为是的渲染与解析，我们可以很幸运的使用其他属性实现几乎一致的效果。  OK，对于不认识display:table-cell属性的IE6/7呢？哦呵呵，很简单，使用display:inline-block属性代替display:table-cell就完全ok的啦！  原因在于：IE6/7下block属性的元素对inline-block属性是有反应，但是却不是纯洁的反应，而是怪蜀黍看到粉嫩小萝莉的一点邪念，就是让元素有个怪异的haslayout属性。//zxx:大家似乎都喜欢用haslayout解析一些老IE下的一些怪异现象，但我自己打心底里是不认同这个概念。  如果IE6/7是很标准的纯洁的解释inline-block属性的话，是无法实现自适应的，右侧的文字描述内容会跑到头像的下面，哦呵呵~~有点负负得正，以毒攻毒的意味。代码如下：  display:table-cell; \*display:inline-block;  就万事大吉，收工回家了。  在本例demo中，右侧内容足够多，所以宽度完整的撑开了，如果内容有限，则宽度就是内容的宽度，此时想要让某个元素（例如关闭按钮）右侧定位就会有问题，解决方法就是定义一个非常宽的宽度，就像上面facebook截图中的CSS属性一样，所以，考虑到各种情况，更健壮耐用的CSS代码应如下：  display:table-cell; \*display:inline-block; width:2000px; \*width:auto;  或者使用：  display:table-cell; width:2000px; \*width:auto; \*zoom:1;  这种两栏的自适应布局，不仅不要分别丈量与计算两列的宽度，连"[页面重构鑫三无准则 之无宽度准则](http://www.zhangxinxu.com/wordpress/?p=1152)"中absolute自适应布局的头像宽度都不需要亮了，可以说是更加懒惰，更加直接的好方法。 | | |
| <**style**>  .box {  width: 60%;  margin: 60px auto 0;  padding: 20px;  background: #f5f5f5;  }  .fix {  zoom: 1;  }  .fix:after, .fix:before {  display: table;  content: **""**;  clear: both;  }  .content {  display: table-cell;  \*display: inline-block;  }  .l {  float: left;  }  .mt5 {  margin-top: 5px;  }  .mr10 {  margin-right: 10px;  } </**style**>  <**div** class="box fix">  <**img** class="l mr10" src="images/1.jpg"/>  <**div** class="content">  <**p**>无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、</**p**>  </**div**> </**div**> | > 左侧为头像，右侧内容自适应。其中头像部分使用了float属性，左浮动，IE8+以及Firefox、Chrome、Opera等现代浏览器右侧使用了display:table-cell属性，结果就自适应了，很简单的代码，很神奇的效果。 | |
| display:table-cell下的等高布局 | table表格中的单元格最大的特点之一就是同一行列表元素都等高。所以，很多时候，我们需要等高布局的时候，就可以借助display:table-cell属性。说到table-cell的布局，不得不说一下"匿名表格元素创建规则"：  CSS2.1表格模型中的元素，可能不会全部包含在除HTML之外的文档语言中。这时，那些"丢失"的元素会被模拟出来，从而使得表格模型能够正常工作。所有的表格元素将会自动在自身周围生成所需的匿名table对象，使其符合table/inline-table、table-row、table- cell的三层嵌套关系。  举个例子吧，如果我们为元素使用"display:table-cell;"属性，而不将其父容器设置为"display:table-row;"属性，浏览器会默认创建出一个表格行，就好像文档中真的存在一个被声明的表格行一样。如果您还不是很理解，可见参见支付宝UED的"[基于display:table的CSS布局](http://ued.alipay.com/wd/2008/10/29/%E5%9F%BA%E4%BA%8Edisplaytable%E7%9A%84css%E5%B8%83%E5%B1%80/" \t "http://www.zhangxinxu.com/wordpress/2010/10/%E6%88%91%E6%89%80%E7%9F%A5%E9%81%93%E7%9A%84%E5%87%A0%E7%A7%8Ddisplaytable-cell%E7%9A%84%E5%BA%94%E7%94%A8/_blank)"一文。//zxx:支付宝今年的招牌广告做得很赞的~~  实现等高布局，毫无疑问，display:table-cell是首选，这就好比鼹鼠，生下来就是为了打洞用的。考虑到匿名创建表格元素的问题，所有table-cell元素外一定要留有一个用来包裹的标签。  于是，我们有类似下面的CSS代码：  .list\_row{display:table-row;}  .list\_cell{display:table-cell; width:30%; padding:1.6%; background-color:#f5f5f5;}/\*中间一个元素背景淡蓝，有别于两边的淡灰色\*/  .list\_center{background-color:#f0f3f9;}  HTML结构代码如下：  <div class="list\_row">  <div class="list\_cell">你一定也有过这种感觉的。...罗兰《寂寞的感觉》</div>  <div class="list\_cell list\_center">作为一个被基阿异捅过...水，四积阴功五读书。</div>  <div class="list\_cell">奔波了一...永远幸福快乐！</div>  </div>  结果在现代浏览器下（如下Firefox3.6下截图）  您可以狠狠地点击这里：[table-cell等高布局demo](http://www.zhangxinxu.com/study/201010/table-cell-equal-height-layout.html" \t "http://www.zhangxinxu.com/wordpress/2010/10/%E6%88%91%E6%89%80%E7%9F%A5%E9%81%93%E7%9A%84%E5%87%A0%E7%A7%8Ddisplaytable-cell%E7%9A%84%E5%BA%94%E7%94%A8/_blank) | | |
| 对于不支持display:table-cell属性的IE6/7浏览器，又当如何解决呢？  我们可以使用"补差等高法"，就是一个一个很大的margin-bottom负值配上一个同样大小的padding-bottom值，本例中为了实例需要，margin-bottom值就百来像素。显然，由于两者原理不同，难免需要用到hack，所以demo完整CSS代码如下：  .list\_row{display:table-row; overflow:hidden;}  .list\_cell{display:table-cell; width:30%; margin-bottom:-100px; padding:1.6%; \*padding-bottom:110px; background-color:#f5f5f5; \*float:left;}  .list\_center{background-color:#f0f3f9;}  认识display:table-cell属性的元素对margin属性不敏感，所以上面margin-bottom属性前没有加\*号。  //zxx:"补差等高法"虽然有效，也有兼容性，但是会带来一些潜在的问题，不宜多用。 | | |
| <**style**>  .box{  display: table;  width: 100%;  }  .list\_row {  display: table-row;  }  .list\_cell {  display: table-cell;  padding: 1.6%;  width: 1%;**/\*具体取值，随意，越小：等分\*/** } </**style**> <**div** class="box">  <**div** class="list\_row">  <**div** class="list\_cell">a a</**div**>  <**div** class="list\_cell list\_center">b</**div**>  <**div** class="list\_cell">c c c c c c c c 无上甚深微妙法、 百千万劫难遭遇、 我今见闻得受持、 愿解如来真实义、</**div**>  </**div**> </**div**> |  | |
| display:table-cell下的列表布局 | 一般这类布局都是使用浮动的。但是浮动布局的不足在于：一是需要清除浮动造成影响；二是不支持不定高列表的浮动。替代浮动布局的方法是有的，如果深究细节以及一些思想，方法还不少。其中有一个方法就是使用display:table-cell。  当然，说句心里话，将display:table-cell属性用在列表元素布局上，有点类似于张飞绣衣服，大马拉小车，优势并没有多大发挥，但是，毕竟也算浮动布局的一个备用替换方案。我的下一篇文章将会详细讲解浮动布局的替换方案，其中table-cell方法可以说是里面最不好的一个方法，本着过渡和热身的目的，这里只简单讲过。  首先是效果：您可以狠狠地点击这里：[display:table-cell下的列表布局](http://www.zhangxinxu.com/study/201010/table-cell-list-flow-layout.html" \t "http://www.zhangxinxu.com/wordpress/2010/10/%E6%88%91%E6%89%80%E7%9F%A5%E9%81%93%E7%9A%84%E5%87%A0%E7%A7%8Ddisplaytable-cell%E7%9A%84%E5%BA%94%E7%94%A8/_blank)  可以看到即使模块高度不一致，也不会产生浮动布局可能产生的错位。  由于table-cell需要每行包裹一个独立的标签，所以，在后台repeat输出的时候，需要两次循环，而是列数是限死的（与浮动布局一样）。对于简单的列表，使用display:table-cell是很难看到什么优势的，但是，如果列表复杂，数据多，内容细，display:table-cell可能会像大S订婚的传闻那样让人震惊。  好，点到为止，就说这些。我的下一篇文章还会讲到此属性的布局的。  更新于2016年8月24日  display:table-cell下的列表布局最适用的场景是：列表个数不固定，但是，无论列表几个，都平分容器空间。什么意思呢？就是如果4个列表，希望每个宽度25%，3个就33.3333%，2个列表希望每个宽度50%。此时，没有比display:table-cell更合适的技术了。  父级设置display:table同时宽度为容器宽度，或者直接width:100%，此时，display:table-cell子元素就会自动等分。？（要么保证内部元素宽度一致，或者设宽度是1%） | | |
| display:table-cell自适应布局下连续单词字符换行 | display:table-cell自适应布局下连续单词字符换行  <http://www.zhangxinxu.com/wordpress/2012/01/display-table-cell-display-table-layout-fixed-word-wrap-break-word/>  display:table-cell自适应布局虽然强大，但是，其有个比较烦人的克星，就是连续单词字符换行的问题。例如sbsbsbsbsbsbsbsbsbsb这样的连续字符。对于一般的元素，很好办的直接：  word-wrap:break-word;  就可以横扫5大洲5大洋，但是，碰到南极洲和大洋洲，就会嗝屁！"大洋洲"就是pre标签，"南极洲"就是td等标签。  而display:table-cell声明的作用就是让元素以td标签的形式呈现，因此，对于连续单词字符，display:table-cell下的自适应布局就会中招（支持display:table-cell的IE8+以及其他现代浏览器下）。如何解决捏？  对于pre标签，辅助：  white-space:pre-wrap;  就可以安心回家过年年了。  但是，对于td类标签呢？  经过我的N多此时与实践，发现可行的方法为 → 对于含有连续单词字符的元素（不能是应用了display:table-cell的框架元素），附加大致如下的CSS代码：  display:table;  width:100%;  table-layout:fixed;  word-wrap:break-word;  其中width属性是必须的，但是width值大小您可以根据实际情况进行设置。个人认为100%基本上就可以受用于所有情况，无需另作修改。  您可能会对上面display:table; + table-layout:fixed;的组合感兴趣，这个我回头有空会专门讲下。否则藏在文章里头，既被人骂啰嗦，又没人看到，太不划算了。您实在有兴趣，可以先去网上查查预热预热。  您可以狠狠地点击这里：[display:table-cell下连续单词字符换行问题修复demo](http://www.zhangxinxu.com/study/201201/display-table-cell-table-layout-fixed-word-wrap-break-word.html" \t "http://www.zhangxinxu.com/wordpress/2012/01/display-table-cell-display-table-layout-fixed-word-wrap-break-word/_blank) | | |
| IMG_256  无处理  应用了{word-wrap:break-word;} IE6/IE7小人得志！  应用了{display:table; width:100%; table-layout:fixed; word-wrap:break-word;} 大家都happy! | | |
|  |  | | |
| float + 负margin 实现自适应布局 | | | |
| 单列定宽单列自适应布局 | <**style** type='text/css'>  .cont, .side {  }   .cont {  width: 100%;  float: left;  }   .main {  margin-right: 200px;  background: #DDDDDD;   height: 300px;  }   .side {  float: left;  margin-left: -200px;  width: 200px;  background: #F7F7F7;   height: 100px;  } </**style**> <**div** class='cont'>  <**div** class='main'>main</**div**> </**div**> <**div** class='side'>side</**div**> | > .side元素的margin-left需要是自身的长度，以保证不会换行显示  > .main元素的margin-right需要不小于.side元素的长度，以保证不会重叠显示； | |
| 左右定宽中间自适应布局 | <**style** type='text/css'>  **\*** {  margin: 0;  padding: 0;  }  .cont, .side, .sub-main {  float: left;  }  .cont {  width: 100%;  }  .main {  margin-right: 200px;  margin-left: 300px;  background: #DDDDDD;   height: 300px;  }  .side {  margin-left: -200px;  width: 200px;  background: #F7F7F7;   height: 200px;  }  .sub-main {  margin-left: -100%;  width: 300px;  background: #F7F7F7;   height: 250px;  } </**style**> <**div** class='cont'>  <**div** class='main'>main</**div**> </**div**> <**div** class='side'>side</**div**> <**div** class='sub-main'>sub-main</**div**> |  | |
| 两列等高自适应布局 | <**style** type='text/css'>  .layout {  overflow: hidden;  }   .cont, .side {  float: left;  padding-bottom: 9999px;  margin-bottom: -9999px;  }   .cont {  width: 100%;  }   .main {  margin-right: 200px;  background: #DDDDDD;   height: 300px;  }   .side {  margin-left: -200px;  width: 200px;  background: #F7F7F7;   height: 100px;  } </**style**> <**div** class='layout'>  <**div** class='cont'>  <**div** class='main'>  main  </**div**>  </**div**>  <**div** class='side'>side</**div**> </**div**> | > 保证两边的背景色相同，不会因高度的不同而相异 | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |