排序算法
网易笔试题目

1、标签的属性，body标签有

属性	值	描述
alink	*rgb(x,x,x)* *#xxxxxx* *colorname*	不赞成使用。请使用样式取代它。 规定文档中活动链接（active link）的的颜色。
background	*URL*	不赞成使用。请使用样式取代它。 规定文档的背景图像。
bgcolor	*rgb(x,x,x)* *#xxxxxx* *colorname*	不赞成使用。请使用样式取代它。 规定文档的背景颜色。
link	*rgb(x,x,x)* *#xxxxxx* *colorname*	不赞成使用。请使用样式取代它。 规定文档中未访问链接的默认颜色。
text	*rgb(x,x,x)* *#xxxxxx* *colorname*	不赞成使用。请使用样式取代它。 规定文档中所有文本的颜色。
vlink	*rgb(x,x,x)* *#xxxxxx* *colorname*	不赞成使用。请使用样式取代它。 规定文档中已被访问链接的颜色。

bgcolor是文档背景颜色，background是文档背景图片
需要熟知每种标签的属性

2、考察的是js中的换行和分号，js可以省略分号，所以return以后就不会再执行后面的代码

3、ol标签是数字；ul是点，如果要去掉点，可以在css中加入list-style-type:none

4、区分类型识别函数：typeof、Object.prototype.toString.call()、constructor、instanceof
typeof：可以有括号也可以没有。可以识别标准类型（Null除外），不能识别具体对象类型（Function除外，Function可以识别）。返回的是小写的，比如typeof ‘ss’＝’string’
Object.prototype.toString.call()：可以识别标准类型以及内置对象类型（如date），不能识别自定义类型。小写。
constructor：可以识别标准类型（Undefined和Null除外），可以识别内置对象

类型，可以识别自定义类型。大写
instance：可以识别内置对象类型，不能识别原始类型（如Number，String），可以识别自定义类型和父类型。大写

z-index值只决定同一父元素中的同级子元素的堆叠顺序

5、css中的position默认值是static。position的用法，一些值的默认值。
position和z-index的关系：z-index仅能作用于定位元素上，position为absolute或者relative。

https://www.cnblogs.com/xcsn/p/4664404.html

6、jquery通过id获得元素应该是#id，如果这里符合的话，应该打印出三次，因为绑定了三次。

7、user是一个函数，将userInfor.getUserName这个函数赋值给了user，再次调用user的时候，没有this这个上下文。

8、null的类型是Null

9、考察的是八进制，0开头的数字，代表是八进制

10、闭包和setTimeout
如果要在一定间隔时间内打印应该：

```
for (var i = 1; i <= 3; i++) {
    // setTimeout((function(a){
    //   console.log(a);
    // })(i),i*1000)
    setTimeout((function(a){
        return function(){console.log(a);}
    })(i),i * 1000)
}
```

因为setTimeout里面应该是个函数，而像题目中那样，函数马上就被调用了。

css属性的注意

1、vertical-align：默认值：baseline，无继承性，只有行内元素支持

值	描述
baseline	默认。元素放置在父元素的基线上。
sub	垂直对齐文本的下标。
super	垂直对齐文本的上标
top	把元素的顶端与行中最高元素的顶端对齐
text-top	把元素的顶端与父元素字体的顶端对齐
middle	把此元素放置在父元素的中部。
bottom	把元素的顶端与行中最低的元素的顶端对齐。
text-bottom	把元素的底端与父元素字体的底端对齐。
length	
%	使用 "line-height" 属性的百分比值来排列此元素。允许使用负值。
inherit	规定应该从父元素继承 vertical-align 属性的值。

在平时做项目的时候，有些时候会发生margin重叠的现象，但这种情况并不是一直都会发生，那到底什么时候会发生margin重叠呢。在网上搜索了一些资料，大致做了一些整理。

首先，会发生margin重叠的肯定是同一个BFC内的块级元素，例如div、ul等，不是块级元素不会发生重叠。（内联元素是不能设置高、行高、内外边距的（内边距可以，外边距左右可以），而且内联元素只能容纳文本或者其他内联元素。）

重叠的情况大致可以分为以下几种：

1、当一个元素出现在另一个元素上面时，第一个元素的下外边距与第二个元素的上外边距会发生合并。

2、当一个元素包含在另一个元素中时（假设没有内边距或边框把外边距分隔开），第一个子元素的上边距会和父元素的上边距合并；最后一个子元素的下边距会和父元素的下边距合并。

3、假设有一个空元素，它有外边距，但是没有边框或填充。在这种情况下，上外边距与下外边距就碰到了一起，它们会发生合并。如果这个外边距遇到另一个元素的外边距，它还会发生合并。

当发生重叠的时候：

1、当两个margin都是正值的时候，取两者的最大值；

2、当 margin 都是负值的时候，取的是其中绝对值较大的，然后，从 0 位置，负向位移；

3、当有正有负的时候，先取出负 margin 中绝对值中最大的，然后，和正 margin 值中最大的 margin 相加。

有时候我们并不希望margin会合并，因为不期望的合并会给页面布局带来混淆。前面说过，会发生合并的是同一个BFC内的块级元素，也就是说属于同一个BFC的两个相邻块级元素的margin会发生重叠。那么什么是BFC，怎么生成BFC。

以下内容大多摘自：http://www.cnblogs.com/dojo-lzz/p/3999013.html。
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。
满足下列CSS声明之一的元素便会生成BFC。
1、根元素
2、float的值不为none
3、overflow的值不为visible （这一条不知道为什么对于上下元素的时候不起作用。）
4、display的值为inline-block、table-cell、table-caption
5、position的值为absolute或fixed

浏览器对于BFC这块区域的约束规则如下：
1、内部的Box会在垂直方向上一个接一个的放置
2、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。）
3、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）
4、BFC的区域不会与float的元素区域重叠（不触发BFC的时候，重叠，文字环绕；触发了BFC，不重叠，紧跟在float元素后面，因为设置了float的元素相当于inline-block，所以会紧跟不会换行）
5、计算BFC的高度时，浮动子元素也参与计算
6、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

以上这些约束就可以解释css中的一些规则了。

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（根本原因在于创建BFC的元素，子浮动元素也会参与其高度计算，即不会产生高度塌陷问题）

为了防止发生margin重叠，只需要把发生重叠的两个元素之一生成BFC就可以了，这只是BFC其中一个作用。BFC在css布局中有很多情况可卡因起作用，例如:
1、对于上下相邻的两个元素，只要把其中一个设置为display:inline-block。按理论来说，将其中一个设置为overflow:hidden，也可以达到消除重叠的效果，结果却没有，不知道为什么。
2、给父元素设置border或者padding，子元素的margin就不会与父元素重叠。例如margin-top。
给父元素设置　overflow: hidden或者display: inline-block或者float: left或者position: absolute，子元素的margin就不会与父元素重叠。
给子元素设置display: inline-block，子元素的margin就不会与父元素重叠。
3、当父元素没有设置宽高，子元素浮动的时候，会使得父元素高度塌陷，这时候只需要给父元素设置overflow: hidden或者display: inline-block或者float: left或者position: absolute。都可以解决这个问题。根本原因在于创建BFC的元素，子浮动元素也会参与其高度计算，即不会产生高度塌陷问题
4、与浮动元素相邻的已生成BFC的元素不能与浮动元素相互覆盖。利用该特性可以作为多栏布局的一种实现方式。overflow: hidden或者display: inline-block或者float: left或者position: absolute。

```
.left{
    background:pink;
    float: left;
    width:180px;
  }
.center{
    background:lightyellow;
```

```
    overflow:hidden;

  }
.right{
    background: lightblue;
    width:180px;
    float:right;
  }
```

最后感谢http://www.cnblogs.com/dojo-lzz/p/3999013.html的作者，学到了很多的知识，嘻嘻。

BFC：
inline是否可以生成BFC，也就是是否可以float、overflow、display、position
✅可以

zoom是IE专用属性，firefox等是不支持的。它的本来作用是设置或检索对象的缩放比例，但这作用几乎用不到。
可以让网页实现IE7中的放大缩小功能。比如你想让你的网页缩小为原来的一半，那么就在body中加入style="zoom:0.5"

设置zoom:1可以在IE6下清除浮动、解决margin导致的重叠等问题。

通常，当浮动子元素导致父元素塌陷的时候，只要给父元素加上overflow: hidden;来解决，但是对于IE不行，需要触发其hasLayout属性才可以。

zoom:1就是IE6 专用的 触发 haslayout 属性的。hasLayout是IE特有的一个属性。很多的IE下的css bug都与其息息相关。在IE中，一个元素要么自己对自身的内容进行计算大小和组织，要么依赖于父元素来计算尺寸和组织内容。当一个元素的hasLayout属性值为true时，它负责对自己和可能的子孙元素进行尺寸计算和定位。
hasLayout对于内联元素也可以有效果，当内联元素的hasLayout为true的时候，可以给这个内联元素设定高度和宽度并得到期望的效果。具体关于hasLayout的知识点，可以另外搜索。

通常，在给低版本的IE做兼容的时候会用到zoom:1。例如，清除浮动的时候，我们会这么写
.clearfix::after{content: ".";clear: both;display: block;visibility: hidden;overflow: hidden;height: 0;*zoom:1}
为了防止低版本的IE浏览器不支持after选择器或者某些属性，在最后加上zoom:1来清除浮动。

为了实现inline-block的兼容的时候，我们会这么写：｛display: inline-block;*display:inline;*zoom:1;｝因为在IE6、IE7下，只有设置在默认显示方式为inline的元素上才会生效。前面说过，当内联元素的hasLayout为true的时候，可以给这个内联元素设定高度和宽度并得到期望的效果，所以这样做可以达到兼容inline-block的效果。
这里还要补充一点，为什么*display:inline;*zoom:1;前面有*，*放在css属性前面，表示这个属性仅仅应用到Internet Explorer 7 以及以下版本。因为Internet Explorer 版本 7 以及以下承认非字母数字（除了下划线）前缀的属性。所以这里，IE7以上的版本作用的是display: inline-block;而在IE7及以下的版本中作用的是display:inline;zoom:1。

z-index:
Z-index属性只能工作于那些被定义了position属性的元素中。这并没有被足够的重视，尤其是对于那些新手。

在某些特定的情况下，关于Z-index 属性的解析会在IE6、IE7以及Firefox2版本中存在一些小小的前后矛盾。
IE中的<select>元素：
IE6中的<select>元素是一个窗口控件，所以它总是出现在层叠顺序的顶部而不会顾及到自然层叠顺序、position属性或者是Z-index。

因父容器（元素）被定位的缘故，IE6/7会错误的对其stacking context进行重置。

盒模型的兼容：

针对这个问题，我们经常使用!important来区分Firefox和IE6.0：

```
#content
{
    width: 414px;
    width: 400px !important;
    padding: 5px; /*只给出一个值，表示上右下左的边距都为5px*/
    border-width: 2px;
}
```

Firefox识别!important，而IE6不识别，且!important的width优先级高，因此FF理解为width: 400px，IE6.0理解为width: 414px，从而显示就相同了。

但是问题出在IE7，IE7.0对!important有了识别能力，但是对盒模型的解析却和IE6.0等一样，从而造成很大的麻烦。也就是说，!important的方法在IE7.0下变得不适用了。

1、避免导致这个问题的情景

用父元素来设置padding和border，子元素只设置width，这样表现就一样了。适用所有情况

2、使用条件注释判断语句 IE5?

```
<!--[if lt IE 6]>
<link rel="stylesheet" type="text/css" href="/css/ie5.css">
<![endif]-->
```

3、使用CSS hacks IE5?

```
#news {
  padding:10px;
  border:1px solid;
  width:250px;
  width:228px;
}
```

所有的浏览器都会看到并理解“width:250px”，但IE 5.*/Win不会读取下面的一行，width:228px，但这行会被其余的浏览器解析。所以最后，IE 5.*/Win得到的width值是250px，其他浏览器得到的是228px。这样，在全部的浏览器中我们的新闻列表的总宽度就一致了。

css hack：

CSS Hack大致有3种表现形式，CSS属性前缀法、选择器前缀法以及IE条件注释法（即HTML头部引用if IE）Hack，实际项目中CSS Hack大部分是针对IE浏览器不同版本之间的表现差异而引入的。

属性前缀法(即类内部Hack)：

例如 IE6能识别下划线"_"和星号" * "，**_width**

IE7能识别星号" * "，但不能识别下划线"_"，***width**

IE6~IE10都认识"\9"， **width:10px;\9**

但firefox前述三个都不能认识。

选择器前缀法(即选择器Hack)：例如 IE6能识别*html .class{}，IE7能识别*+html .class{}或者*:first-child+html .class{}。

IE条件注释法(即HTML条件注释Hack)：针对所有IE(注：IE10+已经不再支持条件注释)： <!--[if IE]>IE浏览器显示的内容 <![endif]-->，针对IE6及以下版本： <!--[if lt IE 6]>只在IE6-显示的内容 <![endif]—>。这类Hack不仅对CSS生效，对写在判断语句里面的所有代码都会生效。

CSS hack方式一：条件注释法

这种方式是IE浏览器专有的Hack方式，微软官方推荐使用的hack方式。举例如下

只在IE下生效

<!--[if IE]>

这段文字只在IE浏览器显示

<![endif]-->

只在IE6下生效

<!--[if IE 6]>

这段文字只在IE6浏览器显示

<![endif]-->
只在IE6以上版本生效

<!--[if gte IE 6]>
这段文字只在IE6以上(包括)版本IE浏览器显示
<![endif]-->
只在IE8上不生效
<!--[if ! IE 8]>
这段文字在非IE8浏览器显示
<![endif]-->
非IE浏览器生效
<!--[if !IE]>
这段文字只在非IE浏览器显示
<![endif]-->

CSS hack方式二：类内属性前缀法
属性前缀法是在CSS样式属性名前加上一些只有特定浏览器才能识别的hack前缀，以达到预期的页面展现效果。

hack	写法	实例	IE6(S)	IE6(Q)	IE7(S)	IE7(Q)	IE8(S)	IE8(Q)	IE9(S)	IE9(Q)	IE10(S)	IE10(Q)
*	*color	青色	Y	Y	Y	Y	N	Y	N	Y	N	Y
+	+color	绿色	Y	Y	Y	Y	N	Y	N	Y	N	Y
-	-color	黄色	Y	Y	N	N	N	N	N	N	N	N
_	_color	蓝色	Y	Y	N	Y	N	Y	N	Y	N	N
#	#color	紫色	Y	Y	Y	Y	N	Y	N	Y	N	Y
\0	color:red\0	红色	N	N	N	N	Y	N	Y	N	Y	N
\9\0	color:red\9\0	粉色	N	N	N	N	N	N	Y	N	Y	N
!important	color:blue !important;color:green;	棕色	N	N	Y	N	Y	N	Y	N	Y	Y

说明：*在标准模式中

看s

“-”减号是IE6专有的hack。

“\9” IE6/IE7/IE8/IE9/IE10都生效

“\0” IE8/IE9/IE10都生效，是IE8/9/10的hack

“\9\0” 只对IE9/IE10生效，是IE9/10的hack

IE浏览器各版本 CSS hack 对照表

CSS hack方式三：选择器前缀法

选择器前缀法是针对一些页面表现不一致或者需要特殊对待的浏览器，在CSS选择器前加上一些只有某些特定浏览器才能识别的前缀进行hack。

目前最常见的是

*html *前缀只对IE6生效
*+html *+前缀只对IE7生效
@media screen\9{...}只对IE6/7生效
@media \0screen {body { background: red; }}只对IE8有效
@media \0screen\,screen\9{body { background: blue; }}只对IE6/7/8有效
@media screen\0 {body { background: green; }} 只对IE8/9/10有效
@media screen and (min-width:0\0) {body { background: gray; }} 只对IE9/10有效
@media screen and (-ms-high-contrast: active), (-ms-high-contrast: none) {body { background: orange; }} 只对IE10有效

清除浮动：

在css布局的时候，经常会用到浮动，有浮动的地方就需要清除浮动，浮动一般有两种情况，一类是浮动元素是父元素的最后一个子元素，另一类就是浮动元素不是父元素的最后一个子元素。

这两类情况各自也有几种不同的清除浮动的方法。

1、浮动元素是父元素的最后一个子元素（这种情况通常发生的是父元素的同级元素发生格式混乱）

1）最简单的方式是触发父元素的BFC，也就是给父元素添加overflow: hidden或者display: inline-block。float: left或者position: absolute也能触发父元素的BFC，但这两种可能会产生不期望的布局变化，具体采用哪一种还是看具体情况。

2）在浮动的元素后面加一个空元素，并给这个空元素加一个类，类里面写上clear:both。

  clear{clear:both}。

3）使用css 的after伪元素，给父元素添加clearfix类，并在css中添加：

.clearfix:after{content: ".";display: block;clear: both;visibility: hidden;overflow: hidden;height: 0}

.clearfix{zoom:1}（兼容低版本的浏览器）

2、浮动元素不是父元素的最后一个子元素（这种情况还包括浮动元素同级的下一个元素发生的格式混乱）

1）在浮动元素的后面一个元素加上一个类，并在类里面写上clear:both。

2）使用css的after伪元素，给浮动元素的后面一个元素添加clearfix类，并在css中添加：

.clearfix:after{content: ".";display: block;clear: both;visibility: hidden;overflow: hidden;height: 0}

.clearfix{zoom:1}（兼容低版本的浏览器）

在本质上，清除浮动的方式可以分为两类，一类是触发父元素的BFC，根本原因在于创建BFC的元素，子浮动元素也会参与其高度计算，就不会产生高度塌陷问题。这类方法通常只能适用于浮动元素是父元素的最后一个子元素，因为它无法保证浮动元素同级的元素跟浮动元素不发生格式混乱。

另一类是使用clear属性。clear的作用是：添加clear:both属性的元素，它的周围不出现浮动元素。（从尝试的例子来看，当周围出现浮动元素的时候，它会自己往后退，将自身排版在浮动元素之后。）这种方式通常就是在浮动元素后面添加一个元素，并给添加的元素设置clear:both属性。其实当使用after伪元素的时候，也是先在后面添加了一个点，再用clear:both来实现清除浮动的。

注意：添加清除浮动相关属性的元素不能设置高度，否则会影响清除。

dispaly:flex在使用时会有兼容性问题，解决方法：

display: -webkit-box; /* Chrome 4+, Safari 3.1, iOS Safari 3.2+ */
display: -moz-box; /* Firefox 17- */
display: -webkit-flex; /* Chrome 21+, Safari 6.1+, iOS Safari 7+, Opera 15/16 */
display: -moz-flex; /* Firefox 18+ */

display: -ms-flexbox; /* IE 10 */
display: flex; /* Chrome 29+, Firefox 22+, IE 11+, Opera 12.1/17/18, Android 4.4+ */
display: box;
display: flexbox;

父元素的属性：
display:flex;
flex-direction:方向
row（默认）｜row－reverse｜column｜column－reverse
row：从左往右
row－reverse：从右往左
column：从上往下
flex-wrap:flex－wrap：换行
nowrap（默认）｜wrap｜wrap－reserve
flex-flow:可以同时设置direction和wrap

justify-content:设置容器主轴方向的对齐方式（类似text－align）
flex－start（默认）｜flex－end｜center｜space－between｜space－around
flex－start：左对齐
flex－end：右对齐
center：居中对齐
space－between：平分间隔弹性盒子元素会平均地分布在行里。如果最左边的剩余空间是负数，或该行只有一个子元素，则该值等效于'flex-start'。在其它情况下，第一个元素的边界与行的主起始位置的边界对齐，同时最后一个元素的边界与行的主结束位置的边距对齐，而剩余的伸缩盒项目则平均分布，并确保两两之间的空白空间相等。
space－around：平分，且收尾也分配弹性盒子元素会平均地分布在行里，两端保留子元素与子元素之间间距大小的一半。如果最左边的剩余空间是负数，或该行只有一个伸缩盒项目，则该值等效于’center'。在其它情况下，伸缩盒项目则平均分布，并确保两两之间的空白空间相等，同时第一个元素前的空间以及最后一个元素后的空间为其他空白空间的一半。

align-items:
设置容器cross－axis方向上的对齐方式 （类似vertical－align）当子元素的高度都不一样的时候

flex－start｜flex－end｜center｜baseline｜stretch（默认）

flex－start：居上对齐

flex－end：居下对齐

center：居中对齐

baseline：基线对齐

stretch：拉伸充满辅轴空间(当给某个元素设置高度以后就不起作用)

align-content:设置cross－axis方向上行对齐方式（多行如何分配在容器中）也就是当换行的时候，容器中有多行，这几行之间怎么分布。

flex－start｜flex－end｜center｜space－between｜space－around｜stretch（默认）

flex-start：各行向弹性盒容器的起始位置堆叠。弹性盒容器中第一行的侧轴起始边界紧靠住该弹性盒容器的侧轴起始边界，之后的每一行都紧靠住前面一行。

flex-end：各行向弹性盒容器的结束位置堆叠。弹性盒容器中最后一行的侧轴起结束界紧靠住该弹性盒容器的侧轴结束边界，之后的每一行都紧靠住前面一行。

center：各行向弹性盒容器的中间位置堆叠。各行两两紧靠住同时在弹性盒容器中居中对齐，保持弹性盒容器的侧轴起始内容边界和第一行之间的距离与该容器的侧轴结束内容边界与第最后一行之间的距离相等。（如果剩下的空间是负数，则各行会向两个方向溢出的相等距离。）

space-between：各行在弹性盒容器中平均分布。如果剩余的空间是负数或弹性盒容器中只有一行，该值等效于'flex-start'。在其它情况下，第一行的侧轴起始边界紧靠住弹性盒容器的侧轴起始内容边界，最后一行的侧轴结束边界紧靠住弹性盒容器的侧轴结束内容边界，剩余的行则按一定方式在弹性盒窗口中排列，以保持两两之间的空间相等。

space-around：各行在弹性盒容器中平均分布，两端保留子元素与子元素之间间距大小的一半。如果剩余的空间是负数或弹性盒容器中只有一行，该值等效于'center'。在其它情况下，各行会按一定方式在弹性盒容器中排列，以保持两两之间的空间相等，同时第一行前面及最后一行后面的空间是其他空间的一半。

stretch：各行将会伸展以占用剩余的空间。如果剩余的空间是负数，该值等效于'flex-start'。在其它情况下，剩余空间被所有行平分，以扩大它们的侧轴尺

寸。

子元素的属性：

flex-basis:设置item初始宽高

main－size（默认）｜<width>（px等）

设置flex item的初始宽（横向排）/ 高（纵向排）

flex-grow:元素所能分到的空余的空间的比例

<number>

initial：0

父元素减去flex-basis后，剩下的空间根据flex-grow按比例进行分配。如果有子元素只设置了width，剩下的元素会减去这个元素的width再根据flex-grow分配；如果一个子元素同时设置了flex-grow和width，这个元素的width会被忽略，直接参与flex-grow的分配；如果一个子元素同时设置了flex-grow和flex-basis，那么这个子元素先被分配flex-basis，再根据flex-grow跟其他元素一起参与分配。

flex:shrink:当元素超过了容器空间

<shrink>

initial：1

将超过容器的空间按照flex:shrink比例分配，各自缩减。当父元素设置了flex-wrap:wrap，shrink就相当于被忽略了，因为不会再进行压缩，而是会换行，且根据flex-grow在每一行进行分配。

flex:可以同时设置basis、grow、shrink，顺序为 <‘flex－grow’> || <‘flex－shrink’> || <‘flex－basis’>

align-self:设置单个flex item在cross－axis方向上的对齐方式

auto（默认：设置在容器上的对齐方式）｜flex－start｜flex－end｜center｜baseline｜

stretch.auto：如果’align-self'的值为'auto'，则其计算值为元素的父元素的'align-items'值，如果其没有父元素，则计算值为'stretch'。

flex-start：弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴起始边界。

flex-end：弹性盒子元素的侧轴（纵轴）起始位置的边界紧靠住该行的侧轴结束边界。

center：弹性盒子元素在该行的侧轴（纵轴）上居中放置。（如果该行的尺寸小于弹性盒子元素的尺寸，则会向两个方向溢出相同的长度）。

baseline：如弹性盒子元素的行内轴与侧轴为同一条，则该值与’flex-start'等效。其它情况下，该值将参与基线对齐。

stretch：如果指定侧轴大小的属性值为'auto'，则其值会使项目的边距盒的尺寸尽可能接近所在行的尺寸，但同时会遵照'min/max-width/height'属性的限制。

transition：

```
a {
  -moz-transition: background 0.5s ease-in,color 0.3s ease-out; //Firefox
  -webkit-transition: background 0.5s ease-in,color 0.3s ease-out; //Chrome、Safari
  -o-transition: background 0.5s ease-in,color 0.3s ease-out; //Opera
```

transition: background 0.5s ease-in,color 0.3s ease-out;// W3C标准
}

transition 属性是一个简写属性，用于设置四个过渡属性：
transition-property：规定设置过渡效果的css属性的名称（none | all | indent）
transition-duration：规定完成过渡效果需要多少秒或毫秒（单位为s或者ms）
transition-timing-function：规定速度效果的速度曲线，允许你根据时间的推进去改变属性值的变换速率
有6个可能值：

1、ease：（逐渐变慢）默认值，ease函数等同于贝塞尔曲线(0.25, 0.1, 0.25, 1.0).
2、linear：（匀速），linear 函数等同于贝塞尔曲线(0.0, 0.0, 1.0, 1.0).
3、ease-in：(加速)，ease-in 函数等同于贝塞尔曲线(0.42, 0, 1.0, 1.0).
4、ease-out：（减速），ease-out 函数等同于贝塞尔曲线(0, 0, 0.58, 1.0).
5、ease-in-out：（加速然后减速），ease-in-out 函数等同于贝塞尔曲线(0.42, 0, 0.58, 1.0)
6、cubic-bezier：（该值允许你去自定义一个时间曲线），特定的cubic-bezier曲线。(x1, y1, x2, y2)四个值特定于曲线上点P1和点P2。所有值需在[0, 1]区域内，否则无效。

transition-delay：定义过渡效果何时开始（单位为s或者ms，默认为0）

transform :

在CSS3中transform主要包括以下几种：旋转rotate、扭曲skew、缩放scale和移动translate以及矩阵变形matrix.
transform： none | <transform-function> [<transform-function>]*
也就是：
transform: rotate | scale | skew | translate |matrix;

none:表示不进么变换；<transform-function>表示一个或多个变换函数，以空格分开；换句话说就是我们同时对一个元素进行transform的多种属性操作，例

如rotate、scale、translate三种，以往我们叠加效果都是用逗号（“，”）隔开，但transform中使用多个属性时却需要有空格隔开,记住了是空格隔开。

```
.rotate:hover{transform: rotate(45deg) scale(0.8,0.8) translate(20px) skew(45deg,15deg);}
```

一、旋转rotate

rotate(<angle>)：通过指定的角度参数对原元素指定一个2D rotation（2D 旋转），需先有transform-origin属性的定义。transform-origin定义的是旋转的基点，其中angle是指旋转角度，如果设置的值为正数表示顺时针旋转，如果设置的值为负数，则表示逆时针旋转。如：transform:rotate(30deg):

二、移动translate

移动translate我们分为三种情况：translate(x,y)水平方向和垂直方向同时移动（也就是X轴和Y轴同时移动）；translateX(x)仅水平方向移动（X轴移动）；translateY(Y)仅垂直方向移动（Y轴移动），具体使用方法如下：

1、translate(<translation-value>[, <translation-value>])：通过矢量[tx, ty]指定一个2D translation，tx 是第一个过渡值参数，ty 是第二个过渡值参数选项。如果 未被提供，则ty以 0 作为其值。也就是translate(x,y),它表示对象进行平移，按照设定的x,y参数值,当值为负数时，反方向移动物体，其基点默认为元素中心点，也可以根据transform-origin进行改变基点。如transform:translate(100px,20px):

2、translateX(<translation-value>)： 通过给定一个X方向上的数目指定一个translation。只向x轴进行移动元素，同样其基点是元素中心点，也可以根据transform-origin改变基点位置。如：transform:translateX(100px):

3、translateY(<translation-value>)：通过给定Y方向的数目指定一个translation。只向Y轴进行移动，基点在元素心点，可以通过transform-origin改变基点位置。如：transform:translateY(20px):

三、缩放scale

缩放scale和移动translate是极其相似，他也具有三种情况：scale(x,y)使元素水平方向和垂直方向同时缩放（也就是X轴和Y轴同时缩放）；scaleX(x)元素仅水平方向缩放（X轴缩放）；scaleY(y)元素仅垂直方向缩放（Y轴缩放），但它们具有相同的缩放中心点和基数，其中心点就是元素的中心位置，缩放基数为1，如果其值大于1元素就放大，反之其值小于1，元素缩小。下面我们具体来看看这三种情况具体使用方法：

1、scale(<number>[, <number>])：提供执行[sx,sy]缩放矢量的两个参数指定一个2D scale（2D缩放）。如果第二个参数未提供，则取与第一个参数一样的值。scale(X,Y)是用于对元素进行缩放，可以通过transform-origin对元素的基点进行设置，同样基点在元素中心位置；基中X表示水平方向缩放的倍数，Y表示垂直方向的缩放倍数，而Y是一个可选参数，如果没有设置Y值，则表示X，Y两个方向的缩放倍数是一样的。并以X为准。如：transform:scale(2,1.5):

scaleX(<number>)： 使用 [sx,1] 缩放矢量执行缩放操作，sx为所需参数。scaleX表示元素只在X轴(水平方向)缩放元素，他的默认值是(1,1)，其基点一样是在元素的中心位置，我们同样是通过transform-origin来改变元素的基点。如：transform:scaleX(2):

3、scaleY(<number>)： 使用 [1,sy] 缩放矢量执行缩放操作，sy为所需参数。scaleY表示元素只在Y轴（垂直方向）缩放元素，其基点同样是在元素中心位置，可以通过transform-origin来改变元素的基点。如transform:scaleY(2):

四、扭曲skew

扭曲skew和translate、scale一样同样具有三种情况：skew(x,y)使元素在水平和垂直方向同时扭曲（X轴和Y轴同时按一定的角度值进行扭曲变形）；skewX(x)仅使元素在水平方向扭曲变形（X轴扭曲变形）；skewY(y)仅使元素在垂直方向扭曲变形（Y轴扭曲变形），具体使用如下：

1、skew(<angle> [, <angle>])： X轴Y轴上的skew transformation（斜切变换）。第一个参数对应X轴，第二个参数对应Y轴。如果第二个参数未提供，则值为0，也就是Y轴方向上无斜切。skew是用来对元素进行扭曲变行，第一个参数是水平方向扭曲角度，第二个参数是垂直方向扭曲角度。其中第二个参数是可选参数，如果没有设置第二个参数，那么Y轴为0deg。同样是以元素中心为基点，我们也可以通过transform-origin来改变元素的基点位置。如：transform:skew(30deg,10deg):

2、skewX(<angle>)： 按给定的角度沿X轴指定一个skew transformation（斜切变换）。skewX是使元素以其中心为基点，并在水平方向（X轴）进行扭曲变行，同样可以通过transform-origin来改变元素的基点。如：transform:skewX(30deg)

3、skewY(<angle>)： 按给定的角度沿Y轴指定一个skew transformation（斜切变换）。skewY是用来设置元素以其中心为基点并按给定的角度在垂直方向（Y轴）扭曲变形。同样我们可以通过transform-origin来改变元素的基点。如：transform:skewY（10deg）

transform-origin(X,Y):用来设置元素的运动的基点（参照点）。默认点是元素的中心点。其中X和Y的值可以是百分值,em,px，其中X也可以是字符参数值left,center,right；Y和X一样除了百分值外还可以设置字符值top,center,bottom，这个看上去有点像我们background-position设置一样；下面我列出他们相对应的写法：

1、top left | left top 等价于 0 0 | 0% 0%

2、top | top center | center top 等价于 50% 0

3、right top | top right 等价于 100% 0

4、left | left center | center left 等价于 0 50% | 0% 50%

5、center | center center 等价于 50% 50%（默认值）

6、right | right center | center right 等价于 100% 50%

7、bottom left | left bottom 等价于 0 100% | 0% 100%

8、bottom | bottom center | center bottom 等价于 50% 100%

9、bottom right | right bottom 等价于 100% 100%

其中 left,center right是水平方向取值，对应的百分值为left=0%;center=50%;right=100%而top center bottom是垂直方向的取值，其中top=0%;center=50%;bottom=100%;如果只取一个值，表示垂直方向值不变，我们分别来看看以下几个实例

浏览器前缀和内核：

前缀

-webkit- // Chrome、Safari

-moz- //Firefox

-ms- //IE

-o- //Opera

内核

Trident：IE、遨游、世界之窗、Avant、腾讯TT、搜狗、360

Gecko：Firefox、Netscape6-9

Webkit：Safari、Chrome

Presto：Opera

ul、ol前面的点和数字的样式：

list-style：

none不使用项目符号

disc实心圆，默认值

circle空心圆

square实心方块

decimal阿拉伯数字

lower-roman小写罗马数字

upper-roman大写罗马数字

lower-alpha小写英文字母

upper-alpha大写英文字母

list-style-type:

list-style-image

CSS宽度塌陷

1、position:absolute

2、position:fixed

3、float

4、父元素设置了flex

5、overflow:hidden

相当于变成了inline-block，宽度跟随内容变化，且可以设置宽度

position:absolute和float:left

1、设置float需要清除浮动，清除浮动的元素，不能设置高度，否则会失效

2、float由于是半脱离文档流，所以后续的元素的内容不会跟浮动元素重复。例如：浮动元素设置了宽度，后续元素的内容会出现在所设置宽度之后。absolute是完全脱离文档流，后续元素会与绝对定位元素完全重合。

3、当浮动元素的宽度大于等于后续元素的宽度的时候，后续元素就会在浮动元素之后，看上去像是没有浮动。（这也是因为内容不能重合的原因，其实后续元素的高度包括了自身内容和浮动元素的高度）

4、有浮动子元素的父元素，可以通过触发BFC来撑起父元素；但是如果子元素是绝对定位就无法撑起。根本原因在于创建BFC的元素，子浮动元素也会参与其高度计算.

display:inline 在IE7极其之前不太兼容

Bootstrap hidden and visible：

在页面中添加样式的方式有：

1、内联式 作为标记引用

2、嵌入式 内部样式表 head内

3、外部引用 link

4、导入样式 @import url(“style.css”）这种方式不能写在页面中，要写在css文件中。括号中的引号可有可无。

import有以下特点：

1，@import url（）机制是不同于link的，link是在加载页面前把css加载完毕，而@import url（）则是读取完文件后在加载，所以会出现一开始没有css样式，闪烁一下出现样式后的页面(网速慢的情况下)。

2，@import 是css2里面的，所以古老的ie5不支持。

3，当使用javascript控制dom去改变样式的时候，只能使用link标签，因为@import不是dom可以控制的。

4，link除了能加载css外还能定义RSS，定义rel连接属性，@import只能加载css

css首字母大写：

text-transform 值：

Capitalize 英文拼音的首字母大写

Uppercase 英文拼音字母全大写

Lowercase 英文拼音字母全小写

响应式布局：

1、viewport

2、@media

css3动画：

animation

选择器的权重：

- !important
- 行内样式，就是写在标签的style属性里面的样式（权重1000）
- ID选择器的数量 （权重100）
- 类、伪类和属性选择器的数量 （权重10）
- 标签选择器和伪元素选择器 （权重1）
- *（匹配所有）

居中的方法：

1、bootstrap的"center-block"类=>margin:0 auto

自己相对于父元素居中

2、center标签 已被废弃

center的子元素居中

3、div中的align:center，例如：

div中的子元素居中

换行的知识：

white-space： 一般对标签设置了宽度，里面的文字默认就会自动换行，这个css属性用于设置是否阻止换行，以及是否保留空白。这里的自动换行是不会在单词内换行的。

normal：空白被浏览器忽略

pre：保留空白

nowrap：不换行，直到遇到

pre-wrap：保留空白，正常换行

pre-line：合并空白，保留换行

inherit

word-break：由于浏览器默认的换行是不会在单词内换行的，所以，这个属性的作用是设置是否允许在单词内换行。

normal

break-all：允许单词内换行

keep-all：仅半角空格、连字符处换行

word-wrap：

normal：不换行，溢出

break-word：边界内换行，允许断词

text-overflow：超出宽度如何处理。

clip：修剪

ellipsis：省略号代替修建文本

string：给定字符串代替修剪文本

截断并加省略号：

text-overflow:ellipsis;

overflow:hidden;

white-space:nowrap;

margin的百分比：基于父元素的宽度。

css媒体查询：

1、link元素中的css媒体查询

```
<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />
```

2、样式表中的css媒体查询：

```
<style>
@media screen and (max-width: 600px) and  (min-width: 500px){
  .facet_sidebar {
    display: none;
  }
}
</style>
```

postion:absolute和relative

absolute：脱离文档流，后面的元素会顶上来

`relative`：保留原来的文档流，其他元素的位置不会改变。

`table:`
`display:table` 表格
`display:tablecell` 单元格
`<table>`

- 单元格边距(表格填充)(cellpadding) -- 代表单元格外面的一个距离,用于隔开单元格与单元格空间单；
- 元格间距(表格间距)(cellspacing) -- 代表表格边框与单元格补白的距离,也是单元格补白之间的距离。
- 边框颜色(bordercolor)
- 边框线的粗细(border)

chrome网页自动填充表单变黄

```
input:-webkit-autofill {
    -webkit-box-shadow: 0 0 0 1000px #fff inset !important;
}
```

或者将autocomplete关闭。autocomplete="off"/"on"

bootstrap 栅格系统：
container：根据屏幕大小设置宽度
row：设置padding、margin
col-md-n：设置百分比
所以嵌套的时候里面不能再写container

Positon的使用

position: static ; 则上边距为(20) px 静态定位 top值无效

position: relative ; 则上边距为(30) px 移动的时候会包括margin

position: absolute ; 则上边距为(30) px 移动的时候会包括margin

position: fixed ; 则上边距为(30) px 固定定位的margin也会生效 移动的时候也会包括margin

position: sticky ; 则上边距为(20) px，页面滚动起来为（10）px，margin会无效；页面没滚动的 时候是静态(static)定位

sticky可以用来解决 banner的滚动问题。也就是当滚动的时候，一开始正常滚动，滚到一定程度banner就粘在屏幕顶部

CSS Sprites

1.简介

CSS Sprites在国内很多人叫css精灵，是一种网页图片应用处理方式。它允许将一个页面涉及到的所有零星图片都包含到一张大图中， 利用CSS的“background-image”，“background- repeat”，“background-position”的组合进行背景定位， 访问页面时避免图片载入缓慢的现象。

2.优点

（1）CSS Sprites能很好地减少网页的http请求，从而大大的提高页面的性能，这是CSS Sprites最大的优点，也是其被广泛传播和应用的主要原因；

（2）CSS Sprites能减少图片的字节；

（3）CSS Sprites解决了网页设计师在图片命名上的困扰，只需对一张集合的图片命名，不需要对每一个小图片进行命名，从而提高了网页制作效率。

（4）CSS Sprites只需要修改一张或少张图片的颜色或样式来改变整个网页的风格。

3.缺点

（1）图片合并麻烦：图片合并时，需要把多张图片有序的合理的合并成一张图片，并留好足够的空间防止版块出现不必要的背景。

（2）图片适应性差：在高分辨的屏幕下自适应页面，若图片不够宽会出现背景断裂。

（3）图片定位繁琐：开发时需要通过工具测量计算每个背景单元的精确位置。

（4）可维护性差：页面背景需要少许改动，可能要修改部分或整张已合并的图片，进而要改动css。在避免改动图片的前提下，又只能（最好）往下追加图片，但这样增加了图片字节。

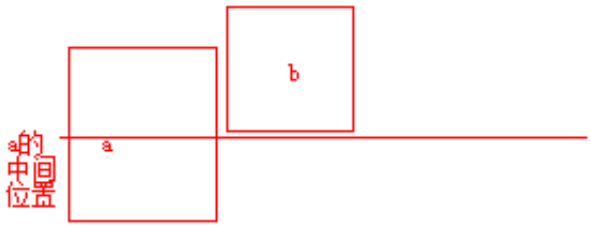
css中的百分比：

margin-top（bottom）或者padding-top（bottom）的百分比取值都是相对于父元素的宽度

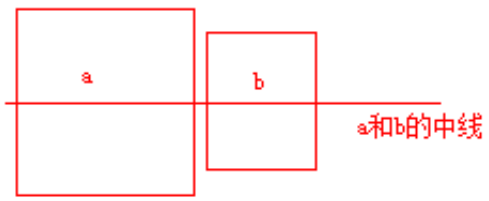
关于vertical-align：

第一种用法，先看后面一句"在表单元格中，这个属性会设置单元格框中的单元格内容的对齐方式。"这很容易理解，如果给一个表格的td加一个vertical-align:middle的样式，表格里面的内容会垂直居中，同样的如果给一个vertical-align:bottom就会底部对齐，如果给一个vertical-align:top就会顶部对齐。

第二种用法，看前页一句"该属性定义行内元素的基线相对于该元素所在行的基线的垂直对齐。"专业的语言我不会说的，可以打个比喻：假设有两个行内元素a和b，a和b都是img，当a加了一个vertical-align:middle样式之后，b的底部（基线）就会对齐a的中间位置，如下图：



如果a和b都加了一个vertical-align:middle样式，那么就互相对齐了对方的中间位置，也就是它们在垂直方向上的中线对齐了，如下图：



在同一行中有两个元素，默认是两个元素低端对齐。当其中一个元素设置了vertical-align:middle，这个元素的中线就会与另一个元素的底端对齐；两个都对齐的话，就会两个元素的中线对齐。而且这两个元素整体都会在它们的父元素内尽量往上靠。
所以要设置某个内联元素在父元素内垂直居中，只要在它后面添加一个元素，并设置为height:100%;width:0;vertival-align:middle;display:inline-block，同时给要居中的元素设置vertical-align:middle。如下：

```
<div style="height: 200px;background-color: red">
    <span style="width: 80px;display: inline-block;vertical-align: middle;">
        Hello Hello Hello Hello Hello Hello Hello
    </span>
    <span style="height: 100%;display: inline-block;width: 0px;vertical-align: middle;background-color: green"></span>
</div>
```

<a>：text-decration:none|underline
：list-style: