TML、DIV+CSS布局，,jquery并掌握CSS在各种浏览器版本下的差异和兼容性

在网站设计的时候，应该注意**css**样式兼容不同浏览器问题，特别是对完全使用**DIV** CSS设计的网，就应该更注意IE6 IE7 FF对CSS样式的兼容，不然，你的网乱可能出去不想出现的效果！

所有浏览器 通用  
height: 100px;

IE6 专用  
\_height: 100px;

IE6 专用  
\*height: 100px;

IE7 专用  
\*+height: 100px;

IE7、FF 共用  
height: 100px !important;  
  
  
一、CSS HACK  
1, !important  
随着IE7对!important的支持, !important 方法现在只针对IE6的HACK.(注意写法.记得该声明位置需要提前.)  
以下为引用的内容：  
<style>  
#wrapper  
{  
width: 100px!important;  
width: 80px;  
}  
</style>

2, IE6/IE7对FireFox  
以下为引用的内容：  
\*+html 与 \*html 是IE特有的标签, firefox 暂不支持.而\*+html 又为 IE7特有标签.  
<style>  
#wrapper  
{  
#wrapper { width: 120px; }  
\*html #wrapper { width: 80px;}  
\*+html #wrapper { width: 60px;}  
}  
</style>

注意:  
\*+html 对IE7的HACK 必须保证**HTML**顶部有如下声明：  
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN”　”http://www.w3.org/TR/html4/loose.dtd”>  
二、万能 float 闭合  
关于 clear float 的原理可参见 [How To Clear Floats Without Structural Markup]  
将以下代码加入Global CSS 中,给需要闭合的div加上 class=”clearfix” 即可,屡试不爽.  
以下为引用的内容：<style>  
.clearfix:after  
{  
content:”.”;  
display:block;  
height:0;  
clear:both;  
visibility:hidden;  
}  
.clearfix  
{  
display:inline-block;  
}  
  
.clearfix {display:block;}  
  
  
</style>

三、其他兼容技巧  
1, FF下给 div 设置 padding 后会导致 width 和 height 增加, 但IE不会.(可用!important解决) 如width:115px !important;width:120px;padding:5px;  
　必须注意的是， !important; 一定要在前面。   
2, 居中问题.  
1).垂直居中.将 line-height 设置为 当前 div 相同的高度, 再通过 vertical-align: middle.( 注意内容不要换行.)  
2).水平居中. margin: 0 auto;(当然不是万能)  
3, 若需给 a 标签内内容加上 样式, 需要设置 display: block;(常见于导航标签)  
4, FF 和 IE 对 BOX 理解的差异导致相差 2px 的还有设为 float的div在ie下 margin加倍等问题.  
5, ul 标签在 FF 下面默认有 list-style 和 padding . 最好事先声明, 以避免不必要的麻烦. (常见于导航标签和内容列表)  
6, 作为外部 wrapper 的 div 不要定死高度, 最好还加上 overflow: hidden.以达到高度自适应.  
7, 关于手形光标. cursor: pointer. 而hand 只适用于 IE.  
兼容代码:兼容最推荐的模式。  
  
.submitbutton {  
float:left;  
width: 40px;  
height: 57px;  
margin-top: 24px;  
margin-right: 12px;  
}  
  
\*html .submitbutton {  
margin-top: 21px;  
}  
  
\*+html .submitbutton {  
margin-top: 21px;  
}

什么是浏览器兼容：当我们使用不同的浏览器（Firefox IE7 IE6）访问同一个网站，或者页面的时候，会出现一些不兼容的问题，有的显示出来正常，有的显示出来不正常，我们在编写CSS的时候会很恼火，刚修复了这个浏览器的问题，结果另外一个浏览器却出了新问题。而兼容就是一种办法，能让你在一个CSS里面独立的写支持不同浏览器的样式。这下就和谐了。呵呵！

最近微软发布的IE7浏览器的兼容性确实给一些网页制作人员添加了一个沉重的负担，虽然IE7已经走向标准化，但还是有许多和FF不同的地方，所以需要用到IE7的兼容。  
有一点逻辑思想的人都会知道可以用IE和FF的兼容结合起来使用，下面介绍三个兼容，例如：（适合新手，呵呵，高手就在这里路过吧。）

程序代码

第一个兼容，IE FF 所有浏览器 公用（其实也不算是兼容）  
height:100px;  
第二个兼容 IE6专用  
\_height:100px;  
第三个兼容 IE6 IE7公用  
\*height:100px;

介绍完了这三个兼容了，下面我们再来看看如何在一个样式里分别给一个属性定义IE6 IE7 FF专用的兼容，看下面的代码，顺序不能错哦：

程序代码

height:100px;  
\*height:120px;  
\_height:150px;

下面我简单解释一下各浏览器怎样理解这三个属性：

在FF下，第2、3个属性FF不认识，所以它读的是 height:100px;

在IE7下，第三个属性IE7不认识，所以它读第1、2个属性，又因为第二个属性覆盖了第一个属性，所以IE7最终读出的是第2个属性 \*height:120px;

在IE6下，三个属性IE6都认识，所以三个属性都可以读取，又因为第三个属性覆盖掉前2个属性，所以IE6最终读取的是第三个属性。

1 针对firefox ie6 ie7的css样式  
现在大部分都是用!important来hack，对于ie6和firefox测试可以正常显示，  
但是ie7对!important可以正确解释，会导致页面没按要求显示！找到一个针  
对IE7不错的hack方式就是使用“\*+html”，现在用IE7浏览一下，应该没有问题了。  
现在写一个CSS可以这样：  
以下为引用的内容：  
#1 { color: #333; }  
\* html #1 { color: #666; }  
\*+html #1 { color: #999; }

那么在firefox下字体颜色显示为#333，IE6下字体颜色显示为#666，IE7下字体颜色显示为#999。  
2 css布局中的居中问题  
主要的样式定义如下：  
body {TEXT-ALIGN: center;}  
#center { MARGIN-RIGHT: auto; MARGIN-LEFT: auto; }  
说明：  
首先在父级元素定义TEXT-ALIGN: center;这个的意思就是在父级元素内的内容居中；对于IE这样设定就已经可以了。  
但在mozilla中不能居中。解决办法就是在子元素定义时候设定时再加上“MARGIN-RIGHT: auto;MARGIN-LEFT: auto; ”  
需要说明的是，如果你想用这个方法使整个页面要居中，建议不要套在一个DIV里，你可以依次拆出多个div，  
只要在每个拆出的div里定义MARGIN-RIGHT: auto;MARGIN-LEFT: auto; 就可以了。  
3 盒模型不同解释  
#box{ width:600px; //for ie6.0- width:500px; //for ff+ie6.0}  
#box{ width:600px!important //for ff width:600px; //for ff+ie6.0 width :500px; //for ie6.0-}  
4 浮动ie产生的双倍距离  
#box{ float:left; width:100px; margin:0 0 0 100px; //这种情况之下IE会产生200px的距离 display:inline; //使浮动忽略}  
这里细说一下block,inline两个元素,Block元素的特点是:总是在新行上开始,高度,宽度,行高,边距都可以控制(块元素);Inline元素的特点是:和其他元素在同一行上,…不可控制(内嵌元素);  
#box{ display:block; //可以为内嵌元素模拟为块元素 display:inline; //实现同一行排列的的效果 diplay:table;  
IE不认得min-这个定义，但实际上它把正常的width和height当作有min的情况来使。这样问题就大了，如果只用宽度和高度，  
正常的浏览器里这两个值就不会变，如果只用min-width和min-height的话，IE下面根本等于没有设置宽度和高度。  
比如要设置背景图片，这个宽度是比较重要的。要解决这个问题，可以这样：  
#box{ width: 80px; height: 35px;}html>body #box{ width: auto; height: auto; min-width: 80px; min-height: 35px;}  
6 页面的最小宽度  
min-width是个非常方便的CSS命令，它可以指定元素最小也不能小于某个宽度，这样就能保证排版一直正确。但IE不认得这个，  
而它实际上把width当做最小宽度来使。为了让这一命令在IE上也能用，可以把一个<div> 放到 <body> 标签下，然后为div指定一个类：  
然后CSS这样设计：  
#container{ min-width: 600px; width:expression\_r\_r(document.body.clientWidth < 600? “600px”: “auto” );}  
第一个min-width是正常的；但第2行的width使用了Javascript，这只有IE才认得，这也会让你的HTML文档不太正规。它实际上通过Javascript的判断来实现最小宽度。  
7 清除浮动  
.hackbox{ display:table; //将对象作为块元素级的表格显示}或者.hackbox{ clear:both;}  
或者加入:after（伪对象）,设置在对象后发生的内容，通常和content配合使用，IE不支持此伪对象，非Ie 浏览器支持，  
所 以并不影响到IE/WIN浏览器。这种的最麻烦的……#box:after{ content: “.”; display: block; height: 0; clear: both; visibility: hidden;}  
8 DIV浮动IE文本产生3象素的bug  
左边对象浮动，右边采用外补丁的左边距来定位，右边对象内的文本会离左边有3px的间距.  
#box{ float:left; width:800px;}#left{ float:left; width:50%;}#right{ width:50%;}\*html #left{ margin-right:-3px; //这句是关键}  
HTML代码<div id=”box”> <div id=”left”></div> <div id=”right”></div></div>  
9 属性选择器(这个不能算是兼容,是隐藏css的一个bug)  
p[id]{}div[id]{}  
这个对于IE6.0和IE6.0以下的版本都隐藏,FF和OPera作用  
属性选择器和子选择器还是有区别的,子选择器的范围从形式来说缩小了,属性选择器的范围比较大,如p[id]中,所有p标签中有id的都是同样式的.  
10 IE捉迷藏的问题  
当div应用复杂的时候每个栏中又有一些链接，DIV等这个时候容易发生捉迷藏的问题。  
有些内容显示不出来，当鼠标选择这个区域是发现内容确实在页面。  
解决办法：对#layout使用line-height属性 或者给#layout使用固定高和宽。页面结构尽量简单。  
11 高度不适应

高度不适应是当内层对象的高度发生变化时外层高度不能自动进行调节，特别是当内层对象使用  
margin 或paddign 时。例：

<div id=”box”>  
<p>p对象中的内容</p>  
</div>

CSS：

#box {background-color:#eee; }  
#box p {margin-top: 20px;margin-bottom: 20px; text-align:center; }

解决方法：在P对象上下各加2个空的div对象CSS代码：.1{height:0px;overflow:hidden;}或者为DIV加上border属性。

屏蔽IE浏览器（也就是IE下不显示）  
\*:lang(zh) select {font:12px !important;}  
select:empty {font:12px !important;}  
这里select是选择符，根据情况更换。第二句是MAC上safari浏览器独有的。

仅IE7识别  
\*+html {…}  
当面临需要只针对IE7做样式的时候就可以采用这个兼容。

IE6及IE6以下识别  
\* html {…}  
这个地方要特别注意很多地主都写了是IE6的兼容其实IE5.x同样可以识别这个兼容。其它浏览器不识别。  
html >body select {……}  
这句与上一句的作用相同。

仅IE6不识别  
select { display :none;}  
这里主要是通过CSS注释分开一个属性与值，流释在冒号前。

仅IE6与IE5不识别  
select { display :none;}  
这里与上面一句不同的是在选择符与花括号之间多了一个CSS注释。

仅IE5不识别  
select { display:none;}  
这一句是在上一句中去掉了属性区的注释。只有IE5不识别

盒模型解决方法  
selct {width:IE5.x宽度; voice-family :""}""; voice-family:inherit; width:正确宽度;}  
盒模型的清除方法不是通过!important来处理的。这点要明确。

清除浮动  
select:after {content:"."; display:block; height:0; clear:both; visibility:hidden;}  
在Firefox中，当子级都为浮动时，那么父级的高度就无法完全的包住整个子级，那么这时用这个清除浮动的兼容来对父级做一次定义，那么就可以解决这个问题。

截字省略号  
select { -o-text-overflow:ellipsis; text-overflow:ellipsis; white-space:nowrapoverflow:hidden; }  
这个是在越出长度后会自行的截掉多出部分的文字，并以省略号结尾，很好的一个技术。只是目前Firefox并不支持。

只有Opera识别  
@media all and (min-width: 0px){ select {……} }  
针对Opera浏览器做单独的设定。

以上都是写CSS中的一些兼容，建议遵循正确的标签嵌套(div ul li 嵌套结构关系)，这样可以减少你使用兼容的频率，不要进入理解误区，并不是一个页面就需要很多的兼容来保持多浏览器兼容)，很多情况下也许一个兼容都不用也可以让浏览器工作得非常好，这些都是用来解决局部的兼容性问题，如果希望把兼容性的内容也分离出来，不妨试一下下面的几种过滤器。这些过滤器有的是写在CSS中通过过滤器导入特别的样式，也有的是写在HTML中的通过条件来链接或是导入需要的补丁样式。

IE5.x的过滤器，只有IE5.x可见  
@media tty {  
i{content:"";}} @import ’ie5win.css’;

IE5/MAC的过滤器，一般用不着

下面是IE的条件注释，个人觉得用条件注释调用相应兼容是比较完美的多浏览器兼容的解决办法。把需要兼容的地方单独放到一个文件里面，当浏览器版本符合的时候就可以调用那个被兼容的样式，这样不仅使用起来非常方便，而且对于制作这个CSS本身来讲，可以更严格的观察到是否有必要使用兼容，很多情况下，当我本人写CSS如果把全部代码包括兼容都写到一个CSS文件的时候的时候会很随意，想怎么兼容就怎么兼容，而你独立出来写的时候，你就会不自觉的考虑是否有必要兼容，是先兼容 CSS？还是先把主CSS里面的东西调整到尽可能的不需要兼容？当你仅用很少的兼容就让很多浏览器很乖很听话的时候，你是不是很有成就感呢？你知道怎么选择了吧～～呵呵

IE的if条件兼容 自己可以灵活使用参看这篇IE条件注释  
Only IE  
所有的IE可识别

只有IE5.0可以识别  
Only IE 5.0+  
IE5.0包换IE5.5都可以识别

仅IE6可识别  
Only IE 7/-  
IE6以及IE6以下的IE5.x都可识别  
Only IE 7/-  
仅IE7可识别

Css 当中有许多的东西不按照某些规律来的话，会让你很心烦，虽然你可以通过很多的兼容，很多的!important 来控制它，但是你会发现长此以往你会很不甘心，看看许多优秀的网站，他们的CSS让IE6,Ie7,Firefox,甚至Safari,Opera运行起来完美无缺是不是很羡慕？而他们看似复杂的模版下面使用的兼容少得可怜。其实你要知道IE 和 Firefox 并不不是那么的不和谐，我们找到一定的方法，是完全可以让他们和谐共处的。不要你认为发现了兼容的办法，你就掌握了一切，我们并不是兼容的奴隶。

div ul li 的嵌套顺序

今天只讲一个规则。就是<div><ul><li>的三角关系。我的经验就是<div>在最外面，里面是<ul>，然后再是<li>，当然<li>里面又可以嵌套<div>什么的，但是并不建议你嵌套很多东西。当你符合这样的规则的时候，那些倒霉的，不听话的间隙就不会在里面出现了，当你仅仅是<div>里面放<li>，而不用<ul>的时候，你会发现你的间隙十分难控制，一般情况下，IE6和IE7会凭空多一些间距。但很多情况你来到下一行，间隙就没了，但是前面的内容又空了很大一块，出现这种情况虽然你可以改变IE的Margin，然后调整Firefox下面的Padding，以便使得两者显示起来得效果很相似，但是你得CSS将变得臭长无比，你不得不多考虑更多可能出现这种问题补救措施，虽然你知道千篇一律来兼容它们，但是你会烦得要命。

具体嵌套写法

遵循上面得嵌套方式，<div><ul><li></li></ul></div> 然后在CSS 里面告诉 ul {Margin:0px;Padding:0px;list-style:none;}，其中list-style:none是不让<li>标记的最前方显示圆点或者数字等目录类型的标记，因为IE和Firefox显示出来默认效果有些不一样。因此这样不需要做任何手脚，你的IE6、和IE7、Firefox显示出来的东西(外距，间距，高度，宽度)就几乎没什么区别了，也许细心的你会在某一个时刻发现：两个象素的差别，但那已经很完美了，不需要你通过调整大片的CSS来控制它们的显示了，你愿意，你可以仅仅兼容一两个地方，而且通常这种兼容可以适应各种地方，不需要你重复在不同的地方调试不同的兼容方式

====各个浏览器中css表现的差异性(浏览器兼容问题)：  
    (http://www.douban.com/group/topic/4629864/)  
    (http://blog.csdn.net/chuyuqing/article/details/37561313)  
    (http://www.iefans.net/ie-9-css-bug/)  
    1,各浏览器下，margin与padding显示差异  
      ==解决办法:CSS reset  
    2,block+float+水平margin,IE6里的间距比超过设置的间距(横向布局)  
      ==解决办法:diaplay:inline  
      (不用担心内联元素无宽高，因为float会让inline元素haslayout,让inline元素表现的和inline-block元素一样有宽高和垂直内外边距)  
      [我们最常用的就是div+CSS布局了，而div就是一个典型的块属性标签，横向布局的时候我们通常都是用div float实现的，横向的间距设置如果用margin实现，这就是一个必然会碰到的兼容性问题]  
    3,inline+(display:block)+float+水平margin,IE6里的间距比超过设置的间距  
      说明:该情况与上面的差不多,只是此处的元素一开始是内联元素,加了display:block的CSS属性.因为它本身就是行内属性标签，所以我们再加上display:inline的话，它的高宽就不可设了。这时候我们还需要在display:inline后面加入display:talbe  
      ==解决办法:display:inline;display:table;  
    4,IE6对margin:0 auto;不会正确的进行解析  
      解决办法：  
         在父元素中使用text-align:center,在元件中使用text-align:left  
    5,无法设置微高(一般小于10px)：  
      说明:当设置元素高度小于10px时，IE6和IE7不受控制，超出设置的高度  
      产生原因:IE不允许原件的高度小于字体的高度  
      解决办法1:设置字体大小:font-size:0;  
      解决办法2:给超出高度的标签设置overflow:hidden  
      解决办法3:设置行高line-height小于你设置的高度  
    6,子元件撑破父元件  
      原因:父元件设置了overflow:auto属性,子元件会从父元件中撑破出来  
      解决办法:父元件中设置position:relative;  
    7,IE无法解析min-height和min-width  
      解决办法1:  
      selector{  
        min-height:150px;  
        height:auto !important;  
        height:150px;  
      }  
      解决办法2：  
      selector{  
        min-height:150px;  
        height:150px;  
      }  
      heml>body selector{  
        height:auto;  
      }  
    8,使用ul li时,li与li之间会空行  
      解决办法1:设置li selector{height:\*\*px;}  
      解决办法2:li selector{float:left;clear:left;}  
      解决办法3:li{display:inline}  
  
====CSS hack:  
    +:IE6,IE7  
    \_:IE6  
    \9:IE6,IE7,IE8  
    \0:IE8,IE9  
    \9\0:IE9  
    !important:All(IE6是有条件的支持)  
  
====常用的CSS reset:  
    (http://blog.bingo929.com/css-reset-collection.html)  
    margin:0;  
    padding:0;  
    border:0;  
    ...          
  
====盒子模型  
    padding,border,margin,content;  
    标准盒模型与IE盒模型：  
       前者：width设置针对content  
       后者:width针对border+padding+content  
    对于盒子的选择：  
       需要依靠最上面的Doctype来看,如果没有声明Doctype,则按照浏览器会根据自己的行为去理解网页;声明后各浏览器会按照W3C标准去解释你的盒子,网页就能在各个浏览器中显示一致了  
  
====性能优化的几个原则:  
      1,尽量减少http请求:  
        改善响应时间的最简单途径就是减少组件的数量，并由此减少HTTP请求的数量:  
        (1)将多个图片合并到一张单独的图片  
        (2)JS、CSS文件合并  
      2,使用内容发布网络(CDN的使用)：  
        是一组分布在多个不同地理位置的Web服务器，用于更加有效地向用户发布内容;  
        主要用于发布页面静态资源：图片、css文件、js文件等。如此，能轻易地提高响应速度.  
      3,添加Expires头  
      4,将CSS样式表放在顶部  
      5,将javascript脚本放在底部  
      6,避免使用CSS表达式  
      7,使用外部javascript和CSS  
      8,减少DNS查询  
      9,精简javascript  
      10,避免重定向  
      11,删除重复脚本  
      12,配置ETag  
      13,使Ajax可缓存  
      14,避免使用CSS表达式  
  
====JS的数据类型(http://blog.sina.com.cn/s/blog\_6fd4b3c10101d0va.html)  
         基本数据类型(5)：string，number，null，undefined，boolean  
         引用类型：object，array，function；  
         数据类型              typeof  
           string             string  
           number              number  
           boolean            boolean  
         undefined           undefined  
           null                object  
           object              object  
           array               object  
          function             function  
          (NaN)                (number)  
          (Error)              (Function)  
    区分基本数据类型：typeof;  
    区分引用数据类型：instanceof（instanceof还可以检测到具体的是什么实例，可以检测是否是正则表达式）  
                   eg:[1,2,3,4] instanceof Array;     //true  
                            \d{3} instanceof RegExp;     //true  
   区分各数据类型：  
            Object.prototype.toString.call(val).slice(8,-1)；  
                eg:Object.prototype.toString.call([1,2,3]).slice(8,-1); //Array  
              obj instanceof typeName;  
                eg:[1,2,3] instanceof Array;     //true  
              obj.constructor.toString().indexof(typeName);  
                eg:[1,2,3].constructor.toString().indexof("Array");      
              //9(只要值不为-1,即为typeName类型)  
  
====会不会SEO(搜索引擎优化)  
    1,站内优化  
    使得网站在搜索引擎上的友好度和站内用户的良好体验度上升  
    让网站在搜索引擎的排名靠前并且得到很好的客户转换率  
    2,站外优化  
    通过SEO手段帮助网站和网站所属企业进行品牌推广  
  
====语义化的理解？  
    html语义化就是让页面的内容结构化，便于对浏览器、搜索引擎解析；  
    在没有样式CSS情况下也以一种文档格式显示，并且是容易阅读的；  
    搜索引擎的爬虫依赖于标记来确定上下文和各个关键字的权重，利于 SEO；  
    使阅读源代码的人对网站更容易将网站分块，便于阅读维护理解。  
  
====html5新特性：  
    1，webStorage(sessionStorage,localStorage);  
    2，onmessage,postmessage解决跨域问题;  
    3，新的文档类型  (New Doctype)：  
      之前的声明文档类型：<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">在H5中失效，H5只需要用  
      <!DOCTYPE html>即可完成文档声明  
    4，新增一系列语义化标签：header,footer,figure+figcaption,article,nav等  
    5，新增属性 ：  
      ==占位符 (placeholder：  
        在HTML4或XHTML中，你需要用JavaScript来给文本框添加占位符。  
        比如，你可以提前设置好一些信息，当用户开始输入时，文本框中的文字就消失。而在HTML5中,新的“placeholder”就简化了这个问题；  
      ==input新增属性：required(必填项);pattern(正则限定输入)；  
    6,audio(音频)与video(视频);  
    7,<script async="async">  
             
====事件模型  
        冒泡型事件(事件委托的原理)  
        捕获型事件   
          
====事件绑定  
    (http://www.cnblogs.com/iloveyoucc/archive/2012/08/15/2639874.html)  
    1,绑定元素属性:事件属性名称由事件类型外加一个on前缀构成  
      eg:<input type="button" name="myButton" onClick="myFunc()">  
      可以支持开发者把参数传递给事件处理器函数  
    2,绑定对象属性:  
      eg:document.forms[0].myButton.onclick = myFunc  
      没有办法向事件函数传递参数  
    3,绑定IE4+支持<script for="id" event="eventName">标识  
      for属性的值必须是元素的id;必须把事件的名称(onmouseover，onclick等等)分配给 event属性  
      eg:<input type="button" name="myButton" id="button1">     
        <script for="button1" event="onclick">  
          // script statements here   
        </script>    
      标识中的语句可以调用页面上其它地方定义的任何函数(或者从.js文件中导入的函数)  
      这种绑定方式意味着您必须为每一个元素和每一个事件创建一个<script for event>标识  
    4,IE5+支持:elemObject.attachEvent("eventName", functionReference);  
      eg:document.getElementById("").attachEvent("onclick",function(){...});  
      注意:不能在元素被载入浏览器之前执行这个语句;  
           该对象的引用在相应的 HTML 按键元素被浏览器创建之前，都是无效的;  
           要让这样的绑定语句或者在页面的底部运行，或者在body元素的onLoad  
           事件处理器调用的函数中运行  
    5,W3C DOM的addEventListener()方法  
      eg:docuemnt.getElementById("").addEventListener("click",function(){...},false);  
      第三个参数表示是否在捕获阶段进行处理  
    【注意方法4和方法5中，事件的名字:IE中要加前缀"on",W3C不用加前缀】  
  
====jsonp&&跨域问题  
  
====如何给函数整个触发器  
    setTimeout()  
    setInterval()  
  
====什么情况下外边距会合并  
    外边距合并指的是，当两个垂直外边距相遇时，它们将形成一个外边距。  
    合并后的外边距的高度等于两个发生合并的外边距的高度中的较大者。  
  
====正则式（http://deerchao.net/tutorials/regex/regex.htm）  
  
====如何理解bfc（http://www.cnblogs.com/lhb25/p/inside-block-formatting-ontext.html）  
  ==BFC布局规则：  
          1，内部的Box会在垂直方向，一个接一个地放置（块级元素独占一行）  
          2，Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠，对于垂直box外边距合并（见上面一条），是因为这两个box属于同一个BFC。因此只要改变其中一个box的BFC即可以达到外边距不重叠的效果  
   (常用的做法是:在其中一个box上套一层容器，并触发该容器生成一个BFC)  
          3，每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。对于一列固定另一栏浮动的两栏自适应布局，固定的一栏会覆盖到浮动的那一栏，即两栏的左边会相接触。根据第四点，可以通过触发固定列的BFC达到自适应的目的。  
          4，BFC的区域不会与float box重叠。  
          5，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的元素。反之也如此因为BFC内部的元素和外部的元素绝对不会互相影响，因此， 当BFC外部存在浮动时，它不应该影响BFC内部Box的布局，BFC会通过变窄，而不与浮动有重叠。同样的，当BFC内部有浮动时，为了不影响外部元素的布局，BFC计算高度时会包括浮动的高度。避免margin重叠也是这样的一个道理。   
          6，计算BFC的高度时，浮动元素也参与计算。对于父元素没有随着子元素高度的增加而增加可以触发父元素的BFC来达到自适应的目的  
  ==哪些元素会生成BFC?  
          根元素  
          float属性不为none  
          position为absolute或fixed  
          display为inline-block, table-cell, table-caption, flex, inline-flex  
          overflow不为visible(所有关于margin，双边距等bug可以用overflow:auto修复)  
  
====ajax  
  
====Http原理  
  
====状态码  
  
====http数据包  
  
====写写自己常用的CSS3特性:  
    ===选择器：  
        ==属性选择器（只有IE6不支持）  
        [att^="value"]  
        匹配包含以特定的值开头的属性的元素  
        eg:div[class^="item"]{background:yellow;}  
        [class以item开头的元素的background属性为yellow]  
        [att$="value"]  
        匹配包含以特定的值结尾的属性的元素  
        [att\*="value"]  
        匹配包含含有特定的值的属性的元素  
        ==连字符（所有浏览器都支持）  
        CSS3中唯一新引入的连字符是通用的兄弟选择器（同级）。它针对一个元素的有同一个父级节点的所有兄弟级别元素。  
        比如，给某个特定的div的同级的图片添加一个灰色的边框(div和图片应该有同一个父级节点)，在样式表中定义下面的样式就足够了：  
        div~img {     
            border: 1px solid #ccc;     
          }  
        ==伪类  
        :nth-child(n)  
        让你基于元素在父节点的子元素的列表位置来指定元素。你可以是用数字、数字表达式或odd 和even 关键词(对斑马样式的列表很完美)。所以如果你想匹配在第四个元素之后的一个3个元素的分组，你可以简单的这样使用：  
        :nth-child(3n+4) { background-color: #ccc; }/\*匹配第4,7,10,13,16,19...个元素\*/  
        :nth-last-child(n)  
        与上个选择器的思想同样，但是从后面匹配元素（倒序），比如，为了指定一个div里面的最后两个段落，我们可以使用下面的选择器：  
        div p:nth-last-child(-n+2)  
        :last-child  
        匹配一个父节点下的最后一个子元素，等同于:nth-last-child(1)  
        :checked  
        匹配选择的元素，比如复选框  
        :empty  
        匹配空元素（没有子元素）。  
        :not(s)  
        匹配所有不符合指定声明(s)的元素。比如，如果你想让所有的没有使用”lead”类的段落的显示为黑色，可以这样写：   
        p:not([class\*="lead"]) { color: black; }  
   ===RGBA和透明度  
   ====多栏布局（火狐浏览器、谷歌浏览器，IE10+等支持）  
      使用多栏布局时只能为所有栏指定一个统一的高度，栏与栏之间的宽度不可能是不一样的，另外也不可能具体指定什么栏中显示什么内容，因此比较适合使用在显示文章内容的时候，不适合用于安排整个网页中由各元素组成的网页结构时。  
      ===多栏布局第一个属性：column-count  
         column-count属性为一个数字表示列数，不带单位，含义是将一个元素中的内容分为多栏进行显示  
         .box {  
          width: 600px;  
          backgrond-color: #ddd;  
          column-count: 2;  
      }  
      ===多栏布局的第二个属性：column-gap  
       使用column-gap属性来设定多栏之间的间隔距离。  
       .box {  
          width: 600px;  
          backgrond-color: #ddd;  
          column-count: 2;  
          column-gap: 60px;  
      }  
    ===多栏布局第3个属性：column-rule  
      column-rule属性在栏与栏之间增加一条间隔线，并且设定该间隔线的宽度、样式、颜色，该属性的指定方法与css中的border属性指定方法相同  
        .box {  
          width: 600px;  
          backgrond-color: #ddd;  
          column-count: 2;  
          column-gap: 60px;  
          column-rule: 5px dashed #000;  
      }  
    ===多栏布局最后一个属性：column-width  
      column-width可以设置每一栏的宽度，但是在实际测试中发现并不像描述的那么简单，遂列举出以下几个问题：  
      在设定column-width的同时必须设置盒子的width，否则盒子宽度默认为100%，每栏宽度按照栏数平均分。  
      盒子每栏宽度必须大于等于column-width设定的值，否则就会减少栏数来增加每栏宽度，例如盒子宽度400px，一共2栏，那么每栏宽度就是200px，如果设置column-width: 210px的话盒子就会变成1栏以保证每栏宽度大于等于column-width:210px;，但是如果每栏宽度大于column-width的值时每栏宽度也不会强制等于column-width，这么看column-width的个性有点像min-width。  
    【CSS3规范里描述的是，各列的高度是均衡的，浏览器会自动调整每列里填充多少文本、均分文本，来使各列的高度保持均衡一致】  
  ===多背景图  
  ===Word Wrap:normal|break-word  
     (http://www.w3chtml.com/css3/properties/text/word-wrap.html)  
     对于文字过长会导致文字撑破容器出来;  
     使用CSS3属性:  
        word-wrap:break-word;内容将在边界内换行,如果需要,单词内部允许断行.  
        word-wrap:normal;内容将会撑破容器盒子  
  ===文字阴影  
  ===@font-face属性  
  ===圆角(边框半径)  
  ===边框图片  
  ===盒阴影  
  ===盒子大小  
  ===媒体查询  
  ===语音  
  
====弹性盒子模型：box-flex  
    CSS3中新的盒子模型——弹性盒子模型(Flexible Box Model)：  
    ==box-flex属性(很适用于流体布局),用来按比例分配父标签的宽度或高度空间  
      eg: #one{box-flex:2;}  
          #two{box-flex:1;}  
          #three{box-flex:1;}  
      表示id分别为one，two，three的元素把父标签按照2:1:1的比例分割;  
    ==父标签需要声明为  
         father{display:box;}  
         father{display:inline-box;}  
         【注意:目前而言,仅Firefox/Chrome/Safari浏览器支持弹性盒子模型(IE9不详,Opera尚未),且使用的时候，需要附带私有前缀。就是诸如-moz-, -webkit-前缀】  
    ==当子元素中有宽度值的时候，此元素就定宽处理，剩下的空间再按比例分配  
      eg：#one{box-flex:2;}  
          #two{box-flex:1;}  
          #three{width:200px;}  
      three宽度为200像素，one和two把剩下的空间按2:1分  
    ==弹性盒子模型下的爸爸(父标签)其实是很有货的。爸爸肚子中的货有：box-orient, box-direction, box-align, box-pack, box-lines. 现在依次讲讲这里box打头的属性都是干嘛用的。  
        ===box-orient用来确定子元素的方向。是横着排还是竖着走：  
          inline-axis是默认值。且horizontal与inline-axis的表现似乎一致的，让子元素横排；而vertical与block-axis的表现也是一致的，让元素纵列  
    =====子元素除了box-flex属性，还有两个属性，box-flex-group和box-ordinal-group，其中box-flex-group的作用不详，貌似目前浏览器也不支持；box-ordinal-group的作用是拉帮结派。数值越小，位置就越靠前，这不难理解，第一组在最前嘛，随后第二组，第三组…   
    例如：box-ordinal-group:1的组就会在box-ordinal-group:2的组前面显示。于是，我们可以利用这个属性改变子元素的顺序。  
    常见浏览器CSS前缀：  
    =====Webkit核心浏览器的(比如, Safari)， 它们以-webkit-开始，以及Gecko核心的浏览器(比如, Firefox)，以-moz-开始，还有Konqueror (-khtml-)、Opera (-o-) 以及Internet Explorer (-ms-)都有它们自己的属性扩展（目前只有IE8支持-ms-前缀）  
  
====设计圆角  
    border-radius:  
  
====客户端存储：  
    localStorage();  
    sessionStorage()  
  
====css选择器权重:  
  style:1000;  
  ID:100;  
  class:10;  
  tagName:1;  
  子选择器(>)全部选择(\*):0;  
  [后面的这些权值只是定的一个相对参考，并不是具体的值]  
  
====link和@import的区别：  
    页面中使用CSS的方式主要有3种：  
         1，行内添加定义style属性值  
         2，页面头部内嵌调用  
         3，外面链接调用其中外面引用有两种：link和@import  
    区别：  
        1：link是XHTML标签，除了加载CSS外，还可以定义RSS等其他事务；  
           @import属于CSS范畴，只能加载CSS。  
        2：link引用CSS时，在页面载入时同时加载；  
           @import需要页面网页完全载入以后加载。  
        3：link是XHTML标签，无兼容问题；  
           @import是在CSS2.1提出的，低版本的浏览器不支持。  
        4：ink支持使用Javascript控制DOM去改变样式；  
           而@import不支持。  
  
====继承的几种方式：  
     1，构造函数  
     2，原型链  
  
====数据传递的几种方式：  
    get,post,本地存储(localStorage)  
    [面试官问我，除了get和post方法还有什么别的方法？我想了半天，面试官说，本地存储啊~~(就是localStorage),其实我想说PUT,HEAD来着@\_@]  
  
====webSocket:  
    [http://www.cnblogs.com/wei2yi/archive/2011/03/23/1992830.html]  
    是下一代客户端-服务器的异步通信方法，该通信取代了单个的TCP套接字，使用ws或wss协议，可用于任意的客户端和服务器程序;而且有一个优秀的第三方API，名为Socket.IO  
    服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息;  
    与ajax的区别:  
    WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；XHR受到域的限制，而WebSocket允许跨域通信  
    // 创建一个Socket实例  
    var socket = new WebSocket('ws://localhost:8080'); //ws表示socket协议  
    // 打开Socket   
    socket.onopen = function(event) {   
      // 发送一个初始化消息  
        socket.send('I am the client and I\'m listening!');   
      // 监听消息  
        socket.onmessage = function(event) {   
          console.log('Client received a message',event);   
        };   
      // 监听Socket的关闭  
        socket.onclose = function(event) {   
          console.log('Client notified socket has closed',event);   
        };   
      // 关闭Socket....   
      socket.close()   
  };  
  
====不用angularJS，如何实现双向数据绑定  
    ajax：异步数据传输？  
  
====响应式布局：弹性盒子/媒体查询  
  
====如何获取浏览器信息：  
    Navigator 对象包含有关浏览器的信息。  
  
Navigator 对象属性：  
  
属性                       描述                IE    F    O  
appCodeName            返回浏览器的代码名。     4    1    9  
appMinorVersion       返回浏览器的次级版本。    4    No   No  
appName               返回浏览器的名称。        4    1    9  
appVersion         返回浏览器的平台和版本信息。 4    1    9  
browserLanguage       返回当前浏览器的语言。    4    No   9  
cookieEnabled      返回指明浏览器中是否启用  
                     cookie 的布尔值。          4    1   9  
cpuClass           返回浏览器系统的 CPU 等级。  4    No  No  
onLine           返回指明系统是否处于脱机模式的布尔值。   
                                                4    No  No  
platform          返回运行浏览器的操作系统平台。4     1   9  
systemLanguage        返回 OS 使用的默认语言。  4     No  No  
userAgent           返回由客户机发送服务器的   
                    user-agent 头部的值。       4     1    9  
userLanguage          返回 OS 的自然语言设置。  4     No  9  
  
====数组操作：  
  ===toString(param):  
    ==null和undefined没有toString()方法  
     把null或undefined变成字符串的方法：null+''或者String(null);  
     直接用(null).toString()会报错。  
    ==param  
    是可选的，当需要把数值型数据转换成相应的进制数时，param可以进行设置；eg:把十进制8转换成二进制：(8).toString(2);//"1000";  
    此处注意：必须是是number类型，若是("8").toString(2);//"8";  
    ==对于浮点数  
      如果小数点后面都是0，调用toString方法会把后面的0去掉  
      (10.00).toString();//"10";(10.01).toString();//"10.01";  
    ==对于object的toString()  
      返回"[object ObjectName]"，其中ObjectName是对象类型的名称。  
    ==对于Date对象：  
      var date = new Date();  
        date.toString();//当前时间信息："Mon Oct 19 2015 19:55:55 GMT+0800 (中国标准时间)"  
    【思考点：如何将小数点后面都为0的浮点数转化为字符串？  
      10.00+"";//"10"(如何得到"10.00");  
      10.00+10.00;//20(为什么不是20.00)；  
      10.01+10.00;//20.009999999999998(为什么不是20.01)  
      】  
    ==String()与toString()的区别：  
      (1)null和undefined有String()转换成字符串，而toString()不能；  
      (2)toString()能设定数值数据转换的进制数，而String()不能；  
      (3)其他情况下：toString(val) === String(val)  
  ===join(seperator)：数组以分隔符seperator转换成字符串  
  
====原生JS对DOM操作：  
    ===selector：  
      ==Document对象方法：  
        document.getElementsByName("");  
        document.getElementsByTagName("");  
        document.getElementById("");  
      ==querySelector()|querySelectorAll()  
       网址:http://www.nowamagic.net/librarys/veda/detail/388  
        (原生写法，比jquery速度快，缺点是IE6、7不支持。)：  
        //接收的参数和CSS选择器接收的参数一样  
        baseElement.querySelector(selector);  
        baseElement.querySelectorAll(selector);  
        baseElement可以是document，也可以是DOM  
        eg:document.querySelectorAll("input[type='checkbox']");  
           document.querySelector === document.querySelectorAll[0];  
        =注意：querySelector与querySelectorAll的区别：  
          (1)querySelector 用来获取一个元素；  
             querySelectorAll 可以获取多个元素。  
          (2)querySelector将返回匹配到的第一个元素，如果没有  
             匹配的元素则返回 NullquerySelectorAll 返回一个包  
             含匹配到的元素的数组，如果没有匹配的元素则返回的数组为空  
      ==document对象与querySelector的区别：  
          (1)getElement方法只能用于document,不能用于DOM,而querySelector可以  
          (2)getElment只能根据name,id,tagName获取DOM，而querySelector不受限制  
      ==关于querySelector的一些bug:  
        <div class= "test" id= "testId" >  
          <p><span>Test</span></p>  
        </div>  
        <script type= "text/javascript" >      
          var   testElement= document.getElementById( 'testId' );   
          var   element = testElement.querySelector( '.test span' );   
          var   elementList = document.querySelectorAll( '.test span' );   
          console.log(element);  // <span>Test</span>  
          console.log(elementList);  // 1     
        </script>  
      按照W3C的来理解，这个例子应该返回：element：null；elementList：[];因为作为baseElement的 testElement里面根本没有符合selectors的匹配子节点；但浏览器却好像无视了baseElement，只在乎selectors，也就是说此时baseElement近乎document；这和我们的预期结果不合.  
      解决办法：看网址里面的内容吧，有点看不懂的感觉  
  
====document.ready与window.onload：  
    (http://blog.sina.com.cn/s/blog\_49fa034d01011lpc.html)  
    Jquery中$(document).ready()的作用类似于传统JavaScript中的window.onload方法，不过与window.onload方法还是有区别的。  
    1.执行时间  
      window.onload必须等到页面内包括图片的所有元素加载完毕后才能执行。  
      $(document).ready()是DOM结构绘制完毕后就执行，不必等到加载完毕。  
    2.编写个数不同  
      window.onload不能同时编写多个，如果有多个window.onload方法，只会执行一个  
      $(document).ready()可以同时编写多个，并且都可以得到执行  
    3.简化写法  
      window.onload没有简化写法  
      $(document).ready(function(){})可以简写成$(function(){});  
    即：document.ready在window.onload前面执行  
  
====document对象：  
    ==每个载入浏览器的HTML文档都会成为Document对象。  
      Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。  
      提示：Document 对象是 Window 对象的一部分，可通过 window.document 属性对其进行访问[window.document可以得到文档的内容(源码)]  
    ==document对象的属性：  
      document.URL:当前页面的url；  
      document.referrer:返回载入当前文档的文档的 URL；  
        (如果当前文档不是通过超级链接访问的，则为null，该属性允许客户端JS  
        访问HTTP引用头部。)  
      document.title:当前页面的title；  
      document.cookie:当前页面所有的cookie；  
      document.lastModified:当前页面最后的修改时间；  
      document.domain:当前页面的域名；  
  
====iframe、frame的缺点：  
  
====html文件的渲染过程(性能优化的依据)：  
    (http://blog.csdn.net/luckygll/article/details/7432713)  
    (http://my.oschina.net/u/1414906/blog/357860)  
    客户端发出请求---服务器响应将html文件返回给请求的客户端浏览器中;  
    页面开始加载;  
    加载从html标签开始;  
    加载到head标签时，如果标签内有个外部样式文件(.css)要加载;  
    客户端向服务器发出一个请求加载CSS文件，服务器响应;  
    CSS加载完成后，客户端浏览器继续加载html文件里的body标签(在CSS文件加载完毕后同时渲染页面);  
    客户端在body标签发现img标签并且引用了一张图片,客户端浏览器向服务器发出一次请求,浏览器不会等到图片下载完，而是继续渲染后面的代码;  
    img标签中的图片加载完了，要显示出来，而图片又要占用一定的面积，又会影响到后面的布局，浏览器不得不回来重新渲染这一部分;  
    body标签里的标签加载及渲染完成后，浏览器发现script标签中的代码,浏览器需要想服务器发出请求加载js文件，服务器响应;  
    浏览器解析执行js文件时发现里面有一些对body文档结构变化的操作(隐藏某段内容等),此时浏览器又需要重新去渲染这些内容;  
    知道浏览器发现 </html>标签;  
    等等，还没完。用户点击了一下界面中的换肤按钮,js让浏览器换了一下<link>的css标签;  
    浏览器召集了在座的各位＜div＞＜span＞＜ul＞＜li＞们，“大伙儿收拾收拾行李，咱得重新来过……”，浏览器向服务器请求了新的CSS文件，重新渲染页面。  
    ===reflow与repaint：  
      ==reflow(回流):涉及到部分页面的布局  
       (http://www.planabc.net/2009/04/13/reflow/)  
       页面中向服务器请求的次数多会影响到打开速度,  
       重新渲染局部的次数也会影响页面打开速度,对于这种来回渲染,叫做reflow,  
       一个元素的回流导致了其所有子元素以及DOM中紧随其后的祖先元素的随后的回流,reflow几乎是无法避免的,我们不能因为害怕reflow就丢掉一些页面效果,但是reflow是可以优化的,  
       (比如图片在.css样式文件中就定义好图片的高度,这会减少页面重新渲染的次数);  
       (1)导致reflow的一些因素:  
          调整窗口大小;  
          改变字体;  
          增加或者移除样式表;  
          内容变化，比如用户在input框中输入文字;  
          激活CSS伪类，比如 :hover (IE 中为兄弟结点伪类的激活);  
          操作class属性;  
          脚本操作DOM;  
          计算offsetWidth和offsetHeight属性;  
          设置style属性的值;  
       (2)合理的优化:  
          如果想设定元素的样式，通过改变元素的class名(尽可能在DOM树的最里层);  
            [可以限制了回流的范围，使其影响尽可能少的节点]  
          避免设置多项内联样式;  
  
          应用元素的动画，使用position属性的fixed值或absolute值  
            [它们不影响其他元素的布局(BFC?),所它他们只会导致重新绘制,而不是一个完整回流]  
          权衡平滑和速度;  
          避免使用table布局;  
          避免使用CSS的JavaScript表达式(仅IE浏览器);  
      ==repaint(重绘):浏览器必须验证DOM树上其他节点元素的可见性  
        重绘：对某个区域、对象的重新渲染表现  
        回流：对某个区域、对象进行重绘，根据条件影响到它的祖先对象进入重绘（并可能无限递归直到顶级祖先对象）  
        重绘如何出现：改变对象的形状、坐标、表现以及内容都会引发该对象被重新渲染，这种现象即为重绘。  
        回流如何出现：当该对象即将重绘时，浏览器会根据条件判断该对象的重绘结果是否会依赖该对象的祖先元素。如果有则将该对象祖先元素也加入本次重绘。并一直向上寻找，直到条件不匹配。此现象即为回流。  
        最后总结：  
          1、重绘可能引发回流  
          2、回流必定引发重绘  
  
    ===js的阻塞特性：(解决:使用异步加载的方式:在script标签中添加async属性)  
      其中js是阻塞式的加载，浏览器在加载js时，当浏览器在执行js代码时，不会做其他的事情，即<script>的每次出现都会让页面等待脚本的解析和执行，js代码执行后，才会继续渲染页面。新一代浏览器虽然支持并行下载。但是js下载仍会阻塞其他资源的下载（比如图片）。所以应该把js放到页面的底部。  
    ===js的优化：  
      1.要使用高效的选择器。  
      2.将选择器保存为局部变量  
      3.先操作再显示  
    
====跨域:  
    协议,域名,端口号有一个不同就被称为跨域  
  
====http与https:  
  
====prototype与\_\_proto\_\_  
    (http://www.cnblogs.com/snandy/archive/2012/09/01/2664134.html)  
    prototype:每一个函数对象都有一个显示的prototype属性,它代表了对象的原型  
    \_\_proto\_\_:内部原型(IE6/7/8/9不支持),每个对象都有一个名为\_\_proto\_\_的内部隐藏属性，指向于它所对应的原型对象,  
    IE9中可以使用Object.getPrototypeOf(obj)获取对象的内部原型;  
    [原型链是基于\_\_proto\_\_才得以形成]  
    所有对象\_\_proto\_\_都指向其构造器的prototype,包括自定义的构造器  
    [注意:构造器可以直接用构造器的名字,也可以用实例对象的constructor属性获得]  
      var obj = {name: 'jack'}  
      var arr = [1,2,3]  
      var reg = /hello/g  
      var date = new Date  
      var err = new Error('exception')  
      //自定义一个构造器  
      function Person(name) {  
        this.name = name  
      }  
      var p = new Person('jack');  
  
      obj.\_\_proto\_\_ === Object.prototype // true  
      obj.\_\_proto\_\_ === obj.constructor.prototype // true  
      arr.\_\_proto\_\_ === Array.prototype  // true  
      reg.\_\_proto\_\_ === RegExp.prototype // true  
      date.\_\_proto\_\_ === Date.prototype  // true  
      err.\_\_proto\_\_ === Error.prototype  // true  
      p.\_\_proto\_\_ === Person.prototype  // true  
      p.\_\_proto\_\_ === p.constructor.prototype  // true  
  
      可以看到p.\_\_proto\_\_与Person.prototype，p.constructor.prototype都是恒等的，即都指向同一个对象。  
  
====作用域与作用域链  
  
====模块化  
  
===跳出循环体  
   break:跳出最内层循环或者退出一个switch语句  
   continue:跳出当前循环继续下一个循环  
   return:跳出循环,即使函数主体中还有其他语句,函数执行也会停止