笔试

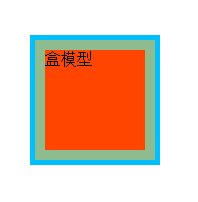
### box-sizing

**理解box-sizing属性border-box，content-box**

**理解box-sizing属性border-box，content-box，其实也是理解正常盒模型与异常盒模型。**

**正常盒模型**

**正常盒模型，是指块元素box-sizing属性为content-box的盒模型。一般在现代浏览器中使用的都是正常盒模型content-box，它也是标准 w3c 盒子模型。**

****

**下图更方便理解。**

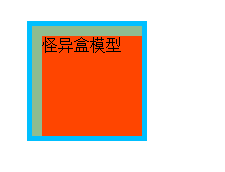
**正常盒模型是指：盒模型的大小柏阔content，padding，border，并且先做content.。**

**正常盒模型的大小会以内容优先自动扩展，内部子元素超过父元素给定的大小，会将父元素撑大。**

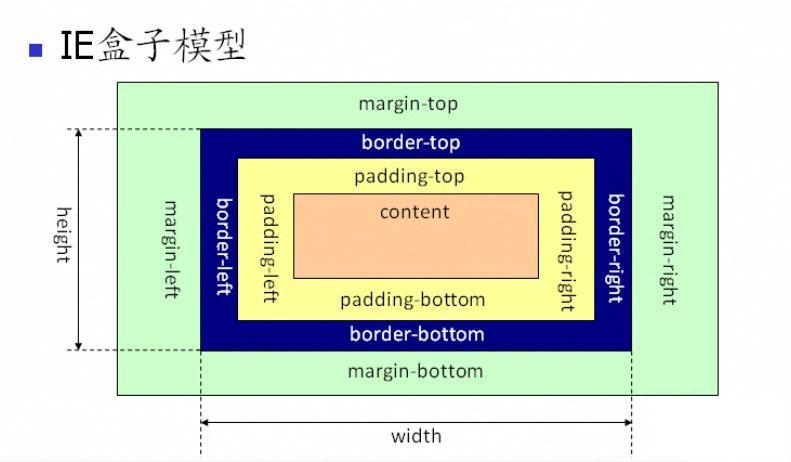


**怪异盒模型**

**怪异盒模型，是指块元素box-sizing属性为border-box的盒模型。一般在IE浏览器中默认为这种怪异盒模型，但是由于其自身的特殊性，手机页面中也有使用怪异盒模型。**



**下图更方便理解。**

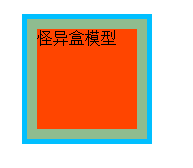


## 怪异盒模型是先做盒。然后添加border，padding，最后做content。即保证盒模型优先，先做盒再放内容，不管内容是否放得下，一般手机上用的更多。

## 更通俗的说，怪异盒模型中，父元素的盒模型确定，子元素是无法撑开父元素的盒模型，只能在盒模型剩余空间展示。

## 如果将父元素盒模型的狂傲，改为（子元素）width + （父元素）padding + （父元素）border的尺寸 = 130px 。 那么得到的效果与正常盒模型相同。

## 如下图所示。



## 那应该选择哪中盒子模型呢？

当然是“标准 w3c 盒子模型”了。

## 怎么样才算是选择了“标准 w3c 盒子模型”呢？

很简单，就是在网页的顶部加上 doctype 声明。假如不加 doctype 声明，那么各个浏览器会根据自己的行为去理解网页，

即 ie 浏览器会采用 ie 盒子模型去解释你的盒子，而 ff 会采用标准 w3c 盒子模型解释你的盒子。所以网页在不同的浏览器中就显示的不一样了。

反之，假如加上了 doctype 声明，那么所有浏览器都会采用标准 w3c 盒子模型去解释你的盒子，网页就能在各个浏览器中显示一致了

[html]

<!doctype html public "-//w3c//dtd xhtml 1.0 transitional//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd">

### Html的块标签

在html中显示模式分为块级和行内，其中常用的块级有：div,p,h1~h6,ul,li,dl,dt,dd...  常用的行内有：span，font,b,u,i,strong,em,a,img,input，其中img和input为行内块元素。

### JavaScript的forEach

forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

注意: forEach() 对于空数组是不会执行回调函数的

|  |  |
| --- | --- |
| 参数 | 描述 |
| function(currentValue, index, arr) | 必需。 数组中每个元素需要调用的函数。 函数参数:   |  |  | | --- | --- | | 参数 | 描述 | | currentValue | 必需。当前元素 | | index | 可选。当前元素的索引值。 | | arr | 可选。当前元素所属的数组对象。 | |
| thisValue | 可选。传递给函数的值一般用 "this" 值。 如果这个参数为空， "undefined" 会传递给 "this" 值 |

列出数组的每个元素：

<button onclick="numbers.forEach(myFunction)">点我</button>

<p id="demo"></p>

<script>

demoP = document.getElementById("demo");

var numbers = [4, 9, 16, 25];

function myFunction(item, index) {

demoP.innerHTML = demoP.innerHTML + "index[" + index + "]: " + item + "<br>"; }

</script>

输出结果：

index[0]: 4  
index[1]: 9  
index[2]: 16  
index[3]: 25

### **Cookie、localStorage、sessionStorage的区别**

浏览器的cookie，localStorage,sessionStorage区别

localStorage,sessionStorage,cookies都是客户端存储的解决方案

1、localStorage和sessionStorage的声明周期

localStorage和sessionStorage都是用来存储客户端临时信息对象，他们只能存储字符串类型的对象，所以我们想要存储其它类型的数据，不得不需要自手动的进行编码和解码，规范中可以存储其它原生类型的对象，目前没有浏览器对其实现。

sessionStorage生命周期为当前窗口或标签页面，它仅仅适用于浏览器会话的持续时间，换句话说当标签或

者窗口关闭的时候，sessionStorage将会被删除。通过sessionStorage存储的数据也就被清空了。

localStorage生命周期为永久有效，除非自己清除了localStorage的内容，不然localStorage存储的信息将永久存在。

2、localStorage和session的共享性

对于不同的浏览器来说，无法共享localStorage或sessionStorage中的信息。

相同浏览器的不同页面间可以共享相同localStorage，前提是页面属于相同的域名和端口（其作用域限定在文档源级别（只要URL的协议、端口、主机名三者中有一个不同，就属于不同的文档源））。

不同的页面或者标签方无法共享sessionStorage的信息，这里需要注意的是，页面及标签页仅仅指顶级窗口，如果一个标签页包含多个iframe标签且他们属于同源页面，那么他们之间是可以共享sessionStorage的。（也就是说如果关闭标签页后，通过sessionStorage存储的数据就都被删除了。sessionStorage的作用域不仅被限制在文档源，还被限定在窗口中，也就是同一标签页中。注意，这里说的窗口是指顶级窗口，若果同一标签页中包含多个<iframe>元素，这两者之间也是可以共享sessionStorage的。）

3、cookie

 cookie的主要内容包括：名字、值、过期时间、路径和域。路径与域一起构成cookie的作用范围。若不设置时间，则表示这个cookie的生命周期为浏览器会话期间，关闭浏览器窗口，cookie就会消失。这种生命周期为浏览器会话期的cookie被称之会话cookie。

 会话cookie一般不存储在硬盘而是保存在内存里，当然这个行为并不是规范规定的。若设置了过期时间，浏览

器就会把cookie保存到硬盘上关闭后再打开浏览器这些cookie仍然有效直到超过设定的过期时间。对于保存在内存

里的cookie，不同的浏览器有不同的处理方式。

4、sessionStorage、localStorage、cookie的区别

 共同点：

都是保存在浏览器端，并且是同源的（URL的协议、端口、主机名是相同的，只要有一个不同就属于不同源）

 不同点：

 1、cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递，而session

Storage和localStorage不会自动把数据发送给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以

限制cookie只属于某个路径下。

 2、存储大小限制也不同，cookie数据不能超过4K，同时因为每次http请求都会携带cookie、所以cookie只适合

保存很小的数据，如会话标识。sessionStorage和localStorage虽然也有存储大小的限制，但比cookie大得多，可以

达到5M或更大

 3、数据有效期不同，sessionStorage仅仅在当前浏览器窗口关闭之前有效；localStorage始终有效，窗口或者

浏览器关闭之后也一直保存，因此作用持久数据；cookie，只在设置cookie过期时间之前有效，即使窗口关闭或者浏

览器关闭。

 4、作用域不同：sessionStorage在不同的浏览器窗口中不共享，即使是同一个页面，localStorage在所有的同源窗

口中是共享的，cookie也是在所有同源的窗口中共享的。

 5、web Storage支持事件通知机制，可以将数据更新的通知发送给监听者。

 6、web Storage的api接口使用更方便。

### Css3和JavaScript写一段动画

https://www.cnblogs.com/zy-plan/p/5222992.html

### 使用过的前端框架、技术和工具

### Slide滚动窗口代码

面试

**ajaxError**  
一般error函数返回的参数有三个： function(jqXHR jqXHR, String textStatus, String errorThrown)。常见调用代码如下：

$.ajax({

url: '/Home/AjaxGetData',

success: function (data) {

alert(data);

},

error: function (jqXHR, textStatus, errorThrown) {

/\*错误信息处理\*/

}

});

这里对这三个参数做详细说明。

第一个参数 jqXHR jqXHR：这里的jqXHR是一个jqXHR对象，在Jquery1.4和1.4版本之前返回的是XMLHttpRequest对象，1.5版本以后则开始使用jqXHR对象，该对象是一个超集，就是该对象不仅包括XMLHttpRequest对象，还包含其他更多的详细属性和信息。

这里主要有4个属性：

readyState :当前状态,0-未初始化，1-正在载入，2-已经载入，3-数据进行交互，4-完成。

status  ：返回的HTTP状态码，比如常见的404,500等错误代码。

statusText ：对应状态码的错误信息，比如404错误信息是not found,500是Internal Server Error。

responseText ：服务器响应返回的文本信息

第二个参数 String textStatus：返回的是字符串类型，表示返回的状态，根据服务器不同的错误可能返回下面这些信息："timeout"（超时）, "error"（错误）, "abort"(中止), "parsererror"（解析错误），还有可能返回空值。

第三个参数 String errorThrown：也是字符串类型，表示服务器抛出返回的错误信息，如果产生的是HTTP错误，那么返回的信息就是HTTP状态码对应的错误信息，比如404的Not Found,500错误的Internal Server Error。

示例代码：

$.ajax({

url: '/AJAX请求的URL',

success: function (data) {

alert(data);

},

error: function (jqXHR, textStatus, errorThrown) {

/\*弹出jqXHR对象的信息\*/

alert(jqXHR.responseText);

alert(jqXHR.status);

alert(jqXHR.readyState);

alert(jqXHR.statusText);

/\*弹出其他两个参数的信息\*/

alert(textStatus);

alert(errorThrown);

}

});

### 闭包

错误的方法

   <!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" xml:lang="en">  
    <head>  
            <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />  
            <title></title>  
    <script>  
    window.onload = function(){  
            var aLi = document.getElementsByTagName('li');  
            for (var i=0;i<aLi.length;i++){  
                    aLi[i].onclick = function(){        //当点击时for循环已经结束  
                    alert(i);  
                    };  
            }  
    }  
    </script>  
              
    </head>  
    <body>  
            <ul>  
                    <li>123</li>  
                    <li>456</li>  
                    <li>789</li>  
                    <li>010</li>  
            </ul>  
    </body>  
    </html>

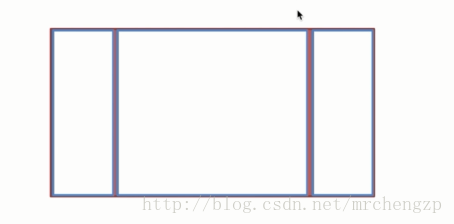
这样的话点击任何一个li标签，都是弹出4

用闭包实现方法

    <!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" xml:lang="en">  
    <head>  
            <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />  
            <title></title>  
    <script>  
    window.onload = function(){  
            var aLi = document.getElementsByTagName('li');  
            for (var i=0;i<aLi.length;i++){  
                    (function(i){  
                            aLi[i].onclick = function(){  
                                    alert(i);  
                            };  
                    })(i);  
            }  
            };  
    </script>  
              
    </head>  
    <body>  
            <ul>  
                    <li>123</li>  
                    <li>456</li>  
                    <li>789</li>  
            </ul>  
    </body>  
    </html>

### 左、中、右布局

# 三栏布局的5种解决方案及优缺点

假设高度已知，请写出三栏布局，左栏、右栏宽度300px，中间宽度自适应。   
   
这道题本身的难度并不大，我们在布局页面的时候，写个三栏布局还是挺简单的。但是如果在面试的时候遇到这道题，就没有那么简单了。看似简单的一道题，想把它答好是不简单的。往往越简单的题越不好答。如果看到这题只想到了浮动和绝对定位，那这题你连及格都及格不了。

下面是5种三栏布局的方法。   
在写布局代码之前，先写两段公共的样式，此段写在头部。

#### 样式

<style media="screen">

html \*{

padding: 0;

margin: 0;

}

.layout article div{

min-height: 100px;

}

</style>

#### 1. 浮动布局

<!--浮动布局 -->

<section class="layout float">

<style media="screen">

.layout.float .left{

float:left;

width:300px;

background: red;

}

.layout.float .center{

background: yellow;

}

.layout.float .right{

float:right;

width:300px;

background: blue;

}

</style>

<h1>三栏布局</h1>

<article class="left-right-center">

<div class="left"></div>

<div class="right"></div>

<div class="center">

<h2>浮动解决方案</h2>

1.这是三栏布局的浮动解决方案；

2.这是三栏布局的浮动解决方案；

</div>

</article>

</section>

浮动布局是有局限性的，浮动元素是脱离文档流，要做清除浮动，这个处理不好的话，会带来很多问题，比如高度塌陷等。   
浮动布局的优点就是比较简单，兼容性也比较好。只要清除浮动做的好，是没有什么问题的。   
延伸：你知道哪些清除浮动的方案？每种方案的有什么优缺点?

#### 2.绝对定位布局

<!-- 绝对布局 -->

<section class="layout absolute">

<style>

.layout.absolute .left-center-right>div{

position: absolute;

}

.layout.absolute .left{

left:0;

width: 300px;

background: red;

}

.layout.absolute .center{

left: 300px;

right: 300px;

background: yellow;

}

.layout.absolute .right{

right:0;

width: 300px;

background: blue;

}

</style>

<h1>三栏布局</h1>

<article class="left-center-right">

<div class="left"></div>

<div class="center">

<h2>绝对定位解决方案</h2>

1.这是三栏布局的绝对定位解决方案；

2.这是三栏布局的绝对定位解决方案;

</div>

<div class="right"></div>

</article>

</section>

绝对定位布局优点，很快捷，设置很方便，而且也不容易出问题，你可以很快的就能想出这种布局方式。   
缺点就是，绝对定位是脱离文档流的，意味着下面的所有子元素也会脱离文档流，这就导致了这种方法的有效性和可使用性是比较差的。

#### 3.flex布局

<!-- flexbox布局 -->

<section class="layout flexbox">

<style>

.layout.flexbox{

margin-top: 110px;

}

.layout.flexbox .left-center-right{

display: flex;

}

.layout.flexbox .left{

# width: 300px;

# background: red;

# }

# .layout.flexbox .center{

# flex:1;

# background: yellow;

# }

# .layout.flexbox .right{

# width: 300px;

# background: blue;

# }

# </style>

# <h1>三栏布局</h1>

# <article class="left-center-right">

# <div class="left"></div>

# <div class="center">

# <h2>flexbox解决方案</h2>

# 1.这是三栏布局的felx解决方案；

# 2.这是三栏布局的flex解决方案；

# </div>

# <div class="right"></div>

# </article>

# </section>

# felxbox布局是css3里新出的一个，它就是为了解决上述两种方式的不足出现的，是比较完美的一个。目前移动端的布局也都是用flexbox。  felxbox的缺点就是不能兼容IE8及以下浏览器。

# 4.表格布局

# <!-- 表格布局 -->

# <section class="layout table">

# <style>

# .layout.table .left-center-right{

# width:100%;

# height: 100px;

# display: table;

# }

# .layout.table .left-center-right>div{

# display: table-cell;

# }

# .layout.table .left{

# width: 300px;

# background: red;

# }

# .layout.table .center{

# background: yellow;

# }

# .layout.table .right{

# width: 300px;

# background: blue;

# }

# </style>

# <h1>三栏布局</h1>

# <article class="left-center-right">

# <div class="left"></div>

# <div class="center">

# <h2>表格布局解决方案</h2>

# 1.这是三栏布局的表格解决方案；

# 2.这是三栏布局的表格解决方案;

# </div>

# <div class="right"></div>

# </article>

# </section>

# 表格布局在历史上遭到很多人的摒弃，说表格布局麻烦，操作比较繁琐，其实这是一种误解，在很多场景中，表格布局还是很适用的，比如这个三栏布局，用表格布局就轻易写出来了。还有表格布局的兼容性很好，在flex布局不兼容的时候，可以尝试表格布局。  表格布局也是有缺陷的，当其中一个单元格高度超出的时候，两侧的单元格也是会跟着一起变高的，而有时候这种效果不是我们想要的。

# 5.网格布局

# <!-- 网格布局 -->

# <section class="layout grid">

# <style>

# .layout.grid .left-center-right{

# width:100%;

# display: grid;

# grid-template-rows: 100px;

# grid-template-columns: 300px auto 300px;

# }

# .layout.grid .left-center-right>div{

# }

# .layout.grid .left{

# width: 300px;

# background: red;

# }

# .layout.grid .center{

# background: yellow;

# }

# .layout.grid .right{

# background: blue;

# }

# </style>

# <h1>三栏布局</h1>

# <article class="left-center-right">

# <div class="left"></div>

# <div class="center">

# <h2>网格布局解决方案</h2>

# 1.这是三栏布局的网格布局解决方案；

# 2.这是三栏布局的网格布局解决方案;

# </div>

# <div class="right"></div>

# </article>

# </section>网格布局也是新出的一种布局方式，如果你答出这种方式，也就证明了你的实力，证明你对技术热点是有追求的，也说明你有很强的学习能力。

# 效果图  IMG_257

# 这五种解决方案应该是最常见的三栏布局，如果你还有其他的方案，欢迎补充！

# 最后这个问题还有很多延伸问题的，比如，

# 高度已知换为高度未知呢？

# 块内内容超出会是怎样的效果？

# 如果是上下高度已知，中间自适应呢？

# 如果是两栏布局呢？

# 如果是上下左右混合布局呢？

# 以上几个延伸你能否轻松应对，如果绝对还有欠缺，尽早查缺补漏吧！

# 欢迎补充！

### Flex布局在浏览器上的区别

# https://blog.csdn.net/qishuixian/article/details/53258807

# <http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html>

### **你知道哪些清除浮动的方案？每种方案的有什么优缺点?**

[8种CSS清除浮动的方法优缺点分析](https://www.cnblogs.com/AnotherLife/p/5800751.html)

为什么清除CSS浮动这么难？

因为浮动会使当前标签产生向上浮的效果，同时会影响到前后标签、父级标签的位置及 width height 属性。  
而且同样的代码，在各种浏览器中显示效果也有可能不相同，这样让清除浮动更难了。

解决浮动引起的问题有多种方法，但有些方法在浏览器兼容性方面还有问题。

我根据自己的经验总结8种清除浮动的方法（测试已通过 ie chrome firefox opera，后面三种方法只做了解就可以了）：

1、父级div定义height

<style type="text/css">

.div1{background:#000080;border:1px solid red;/\*解决代码\*/height:200px;}

.div2{background:#800080;border:1px solid red;height:100px;margin-top:10px}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：父级div手动定义height，就解决了父级div无法自动获取到高度的问题

优点：简单，代码少，容易掌握

缺点：只适合高度固定的布局，要给出精确的高度，如果高度和父级div不一样时，会产生问题

建议：不推荐使用，只建议高度固定的布局时使用

评分：★★☆☆☆

2、结尾处加空div标签clear:both

<style type="text/css">

.div1{background:#000080;border:1px solid red}

.div2{background:#800080;border:1px solid red;height:100px;margin-top:10px}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

/\*清除浮动代码\*/

.clearfloat{clear:both}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

<div class="clearfloat"></div>

</div>

<div class="div2">

div2

</div>

原理：添加一个空div，利用css提高的clear:both清除浮动，让父级div能自动获取到高度

优点：简单，代码少，浏览器支持好，不容易出现怪问题

缺点：不少初学者不理解原理；如果页面浮动布局多，就要增加很多空div，让人感觉很不爽

建议：不推荐使用，但此方法是以前主要使用的一种清除浮动方法

评分：★★★☆☆

3、父级div定义伪类:after和zoom

<style type="text/css">

.div1{background:#000080;border:1px solid red;}

.div2{background:#800080;border:1px solid red;height:100px;margin-top:10px}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

/\*清除浮动代码\*/

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

.clearfloat{zoom:1}

</style>

<div class="div1 clearfloat">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：IE8以上和非IE浏览器才支持:after，原理和方法2有点类似，zoom(IE转有属性)可解决ie6,ie7浮动问题

优点：浏览器支持好，不容易出现怪问题（目前：大型网站都有使用，如：腾迅，网易，新浪等等）

缺点：代码多，不少初学者不理解原理，要两句代码结合使用，才能让主流浏览器都支持

建议：推荐使用，建议定义公共类，以减少CSS代码

评分：★★★★☆

4、父级div定义overflow:hidden

<style type="text/css">

.div1{background:#000080;border:1px solid red;/\*解决代码\*/width:98%;overflow:hidden}

.div2{background:#800080;border:1px solid red;height:100px;margin-top:10px;width:98%}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：必须定义width或zoom:1，同时不能定义height，使用overflow:hidden时，浏览器会自动检查浮动区域的高度

优点：简单，代码少，浏览器支持好

缺点：不能和position配合使用，因为超出的尺寸的会被隐藏

建议：只推荐没有使用position或对overflow:hidden理解比较深的朋友使用

评分：★★★☆☆

5、父级div定义overflow:auto

<style type="text/css">

.div1{background:#000080;border:1px solid red;/\*解决代码\*/width:98%;overflow:auto}

.div2{background:#800080;border:1px solid red;height:100px;margin-top:10px;width:98%}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：必须定义width或zoom:1，同时不能定义height，使用overflow:auto时，浏览器会自动检查浮动区域的高度

优点：简单，代码少，浏览器支持好

缺点：内部宽高超过父级div时，会出现滚动条。

建议：不推荐使用，如果你需要出现滚动条或者确保你的代码不会出现滚动条就使用吧。

评分：★★☆☆☆

6、父级div也一起浮动

<style type="text/css">

.div1{background:#000080;border:1px solid red;/\*解决代码\*/width:98%;margin-bottom:10px;float:left}

.div2{background:#800080;border:1px solid red;height:100px;width:98%;/\*解决代码\*/clear:both}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：所有代码一起浮动，就变成了一个整体

优点：没有优点

缺点：会产生新的浮动问题。

建议：不推荐使用，只作了解。

评分：★☆☆☆☆

7、父级div定义display:table

<style type="text/css">

.div1{background:#000080;border:1px solid red;

/\*解决代码\*/width:98%;display:table;margin-bottom:10px;}

.div2{background:#800080;border:1px solid red;height:100px;width:98%;}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

</div>

<div class="div2">

div2

</div>

原理：将div属性变成表格

优点：没有优点

缺点：会产生新的未知问题

建议：不推荐使用，只作了解

评分：★☆☆☆☆

8、结尾处加br标签clear:both

<style type="text/css">

.div1{background:#000080;border:1px solid red;margin-bottom:10px;zoom:1}

.div2{background:#800080;border:1px solid red;height:100px}

.left{float:left;width:20%;height:200px;background:#DDD}

.right{float:right;width:30%;height:80px;background:#DDD}

.clearfloat{clear:both}

</style>

<div class="div1">

<div class="left">Left</div>

<div class="right">Right</div>

<br class="clearfloat" />

</div>

<div class="div2">

div2

</div>

原理：父级div定义zoom:1来解决IE浮动问题，结尾处加br标签clear:both

建议：不推荐使用，只作了解

评分：★☆☆☆☆