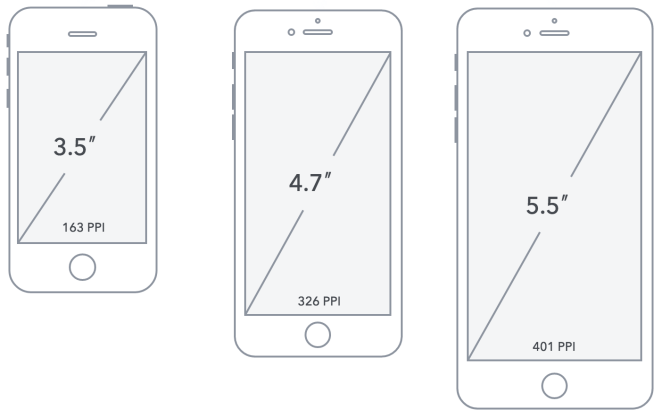
# 移动Web

一、基础知识

1、屏幕

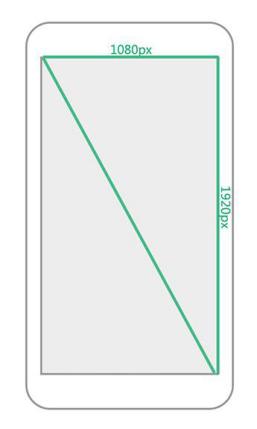
移动设备与PC设备最大的差异在于屏幕，这主要体现在屏幕**尺寸**和屏幕**分辨率**两个方面。

通常我们所指的屏幕尺寸，实际上指的是**屏幕对角线的长度**（一般用**英寸**来度量），并不是指宽高。下图所示：



（1）分辨率

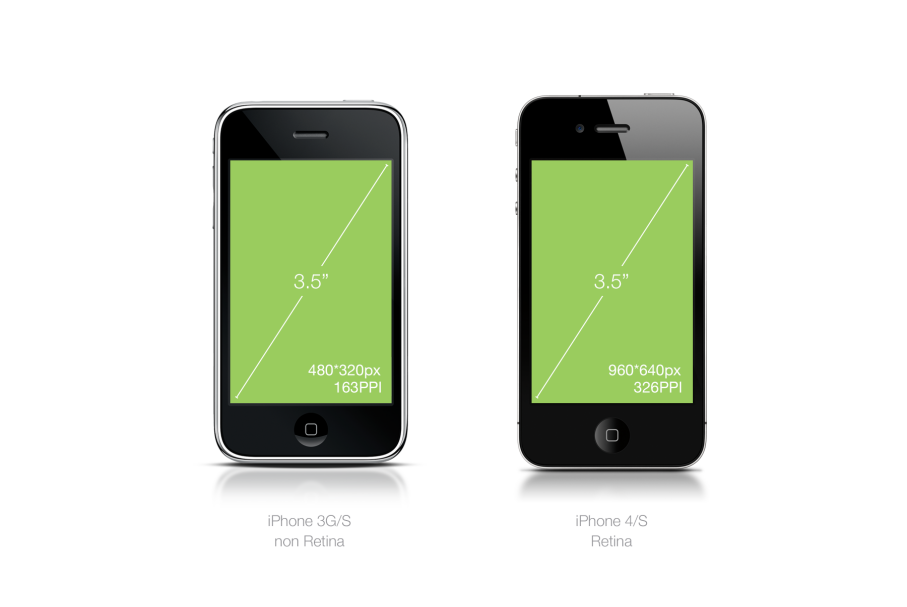
分辨率则一般用像素来度量 px，表示屏幕水平和垂直方向的像素数量，例如1920\*1080指的是屏幕**垂直**方向和**水平**方向分别有1920和1080个像素点而构成（我们一般会认为先宽后高，但其实不是的，这个要注意），如下图所示



2、长度单位

在Web开发中可以使用**px**（像素）、**em**（相对于父容器设置的大小）、**pt**（点，用于ios比较多，ios是以这个为单位的）、**in**（英寸）、**cm**（厘米）做为长度单位，我们最常用px（像素）做为长度单位。

我们可以将上述的几种长度单位划分成**相对长度**单位和**绝对**长度（不管你是在什么设备上面，什么系统下面，这个单位所代表的值，宽高值是不变的，例如1英寸、1厘米，不管在哪里，他就是那么大）单位。



如上图所示，iPhone3G/S和iPhone4/S的屏幕尺寸都为3.5英寸（in）但是屏幕分辨率却分别为480\*320px、960\*480px。

同一个屏幕下，3S中的px会大点，而4S的px会小点，由此我们可以得出英寸是一个绝对长度单位，而像素是一个相对长度单位（像素并没有固定的长度）。

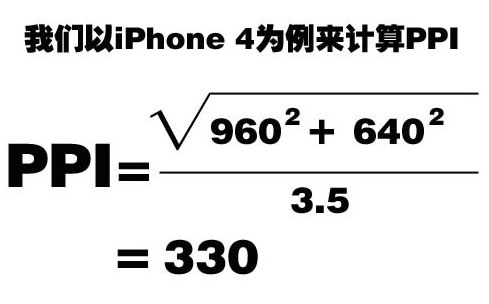
3、像素密度

（1）DPI、PPI

DPI（Dots Per Inch）是印刷行业中用来表示打印机**每英寸可以喷的墨汁点数（一英寸中点的数量）**，计算机显示设备从打印机中借鉴了DPI的概念，由于计算机显示设备中的最小单位不是墨汁点而是像素，所以用PPI（Pixels Per Inch）值来表示屏幕每英寸的像素数量（就是**一英寸中像素的数量**），我们将PPI、DPI都称为像素密度，但PPI应用更广泛，DPI在Android设备比较常见。

（2）计算PPI

如下图所示，利用 勾股定理 我们可以计算得出PPI。



PPI值的越大说明单位尺寸里所能容纳的像素数量就越多，所能展现画面的品质也就越精细，反之就越粗糙。

Retina即视网膜屏幕（IOS的），苹果注册的命名方式，意指具有较高PPI（大于320）的屏幕。

思考：在屏幕尺寸（英寸）固定时，PPI和像素大小的关系？

**结论**：屏幕尺寸固定时，当PPI 越大，像素的实际大小就会越小，当PPI越小，像素实际大小就越大。

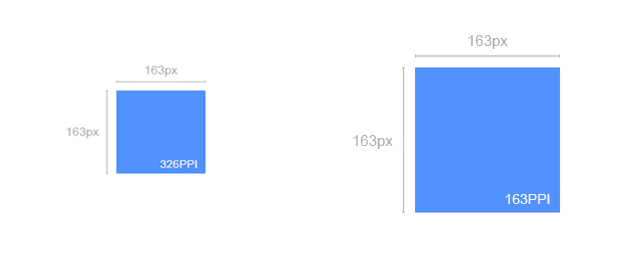
个人理解：屏幕尺寸固定时，当像素的实际大小越小，PPI 越大。当PPI像素实际大小越大越小。结合PPI的定义来理解，你px越小，1英寸肯定装得多。

这样的话一张图在3S和4S中所占的大小在视觉是不一样的，如100px宽高的图片，在3S中会比较大，在4S中会比较小，因为它们的**1px长度不一样**。为了让它们看起来是一样大的，这样引出了设备独立像素，这个东西。

4、设备独立像素

随着技术发展，设备不断更新，出现了不同PPI的屏幕共存的状态（如iPhone3G/S为163PPI，iPhone4/S为326PPI），像素不再是统一的度量单位，**这会造成同样尺寸的图像在不同PPI设备上的显示大小不一样。**

如下图，假设你设计了一个163\*163的蓝色方块，在PPI为163的屏幕上，那这个方块看起来正好就是1\*1英寸大小，在PPI为326的屏幕上，这个方块看起来就只有0.5\*0.5英寸大小了。



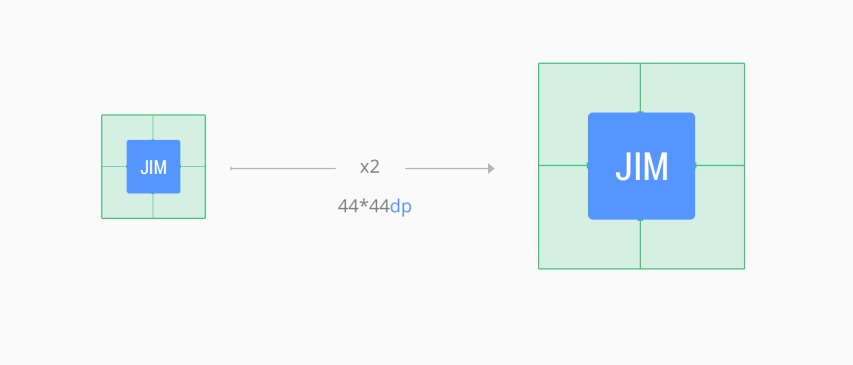
（1）DP、PT

做为用户是不会关心这些细节的，他们只是希望在不同PPI的设备上看到的图像内容差不多大小，所以这时我们需要一个新的单位，**这个新的单位能够保证图像内容在不同的PPI设备看上去大小应该差不多，这就是独立像素**（它是一个单位，但是它不是一个具体的值，这个值可以通过系统来进行得到，后续我们可以根据这个值来做相应的处理）。

在**IOS设备上叫PT(Point)**，**Android**设备上叫DIP(Device independent Pixel)或**DP**。

（2）举例

iPhone 3G（PPI为163）1dp = 1px，iPhone 4（PPI为326）1dp = 2px。



我们也不难发现，如果想要iPhone 3G/S和iPhone 4/S图像内容显示一致，可以把iPhone 4/S的尺寸放大一倍（它们是一个2倍的关系），即在iPhone3G/S的上尺寸为44\*44px，在iPhone4/S上为88\*88px，我们要想实现这样的结果可以设置**44\*44dp**，这时在iPhone3G/S上代表44\*44px，在iPhone4/S上代表88\*88px，最终用可以看到的图像差不多大小。

通过上面例子我们不难发现dp同px是有一个对应（比例）关系的，这个对应（比例）关系是操作系统确定并处理，目的是确保不同PPI屏幕所能显示的图像大小是一致的，通过window.**devicePixelRatio**可以获得该比例值。（举例：img大小被设置为100px，如果没有dp，那么它在不同分辨率的屏幕下，大小会不一样。）

|  |
| --- |
| <script type="text/javascript">  alert(window.devicePixelRatio);  </script> |
| 1  2 |

但是这样会造成图片的失真，把图片放大了。采用的是2倍图

（3）2倍图，理解dp

这个2倍图是美工给的，还有多倍图等来进行适配。

|  |
| --- |
| <style type="text/css">  body {  background-color: black;  color: #fff;  }  img {  width: 57px;// 就是这个57dp的宽度就是网页上57px的宽度。在移动端也要显示这么宽的宽度。  }  </style>  <body>  2倍图:<br /><img src="images/big.png" title="大图片"><br />  原图：<br /><img src="images/small.png" title="小图片">  <script type="text/javascript">  console.log(window.devicePixelRatio);  </script>  </body>  1  原图57px：它要显示 57dp需要114px，它的原图只有57px，图被放大了，有些失真了  2倍图114px：它要显示 57dp需要114px，它的原图有114px，就刚好。相比较会清晰 |

从上图我们得知dp（或pt）和px并不总是绝对的倍数关系（并不总能保证能够整除），而是window.devicePixelRatio = 物理像素 / 独立像素（逻辑像素），然而这其中的细节我们不必关心，因为操作系统会自动帮我们处理好（**保证1dp在不同的设备上看上去大小差不多**）。

（4）

5、像素

（1）物理像素

指的是屏幕渲染图像的最小单位，属于屏幕的物理属性，不可人为进行改变，其值大小决定了屏幕渲染图像的品质，我们以上所讨论的都指的是物理像素。

获取屏幕的物理像素尺寸？？？有问题这里

window.screen.width;

window.screen.height;

// 部分移动设备下获取会有错误，与移动开发无关，只需要了解

|  |
| --- |
| <script type="text/javascript">  var physicsWidth = window.screen.width;  var physicsHeight = window.screen.height;  console.log('物理宽度：', physicsWidth);  console.log('物理高度：', physicsHeight);  </script> |
| 笔记本屏幕的分辨率是 1920\*1080，宽是1920 高1080，跟手机分辨率1440\*720，宽是720，高1440，还是有区别的  1 |

（2）CSS像素，与设备无关像素

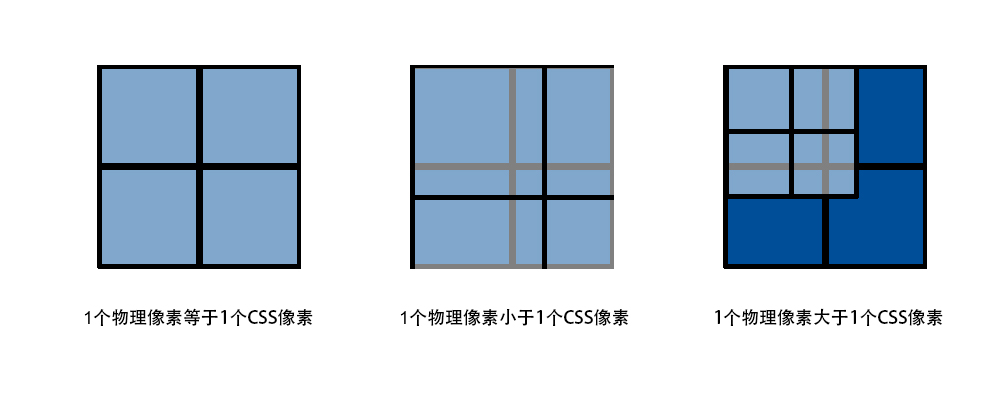
指的是通过CSS进行网页布局时用到的单位，其默认值(PC端)是和物理像素保持一致的（1个单位的CSS像素等于1个单位的物理像素），但是我们可通缩放“ctrl+加号”来改变其大小。

|  |
| --- |
| <style type="text/css">  \* {  padding: 0;  margin: 0;  }  .box {  width: 1536px;  height: 100px;  background-color: skyblue;  }  </style> |
| 1  2 |

（3）CSS像素与物理像素之间的对应关系

我们通过调整浏览器的缩放比例可以直观的理解CSS像素与物理像素之间的对应关系，如下图所示：

注：黑色的是CSS像素



我们需要理解的是物理像素和CSS像素的一个关系，1个物理像素并不总是等于一个CSS像素，通过调整浏览器缩放比例，可以有以上3种情况。

二、\*上述概念的总结(重要)

1、分辨率

物理分辨率和分辨率是2个概念，可别混淆。

（1）物理分辨率

PC端

显示屏是由一个个物理像素点组成的，每个像素可以根据操作系统设置自己的颜色和亮度，屏幕从工厂出来那天起，它上面的物理像素点就固定不变了，除非把屏幕砍掉一块。

**物理分辨率**（标准分辨率）：显示屏的最佳分辨率，即屏幕实际存在的**像素行数**乘以**列数**的数学表达方式，是显示屏固有的参数，**不能调节**，其含义是指显示屏最高可显示的像素数。

**物理分辨率**即LED液晶板的实际分辨率，在LED液晶板上通过网格来划分液晶体，一个液晶体为一个像素点（物理像素）。（我的显示屏为例水平的液晶体（像素点）有 1920个，垂直方向的液晶体（像素点）有 1080个）。

通常用物理分辨率来评价LED显示屏的档次。



我的显示屏的物理分辨率（显示屏最高可显示的像素数）是1920×1080，当前设置分辨率的是1920x1080，分辨率是可以调的。

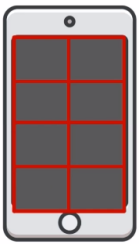
注：移动端，如手机物理分辨率1440x720，指的是垂直方向 1440 个像素点，水平方向 720 个像素点，有的不支持调分辨率。

1. 分辨率

**分辨率是可以改变的。**

分辨率为1600×900 时，就是指在LED液晶板的横向上划分了1600个物理像素点，竖向上划分了900个物理像素点。如果你换成1920×1080，那么LED液晶板就会横纵向重新划分物理像素点。 物理分辨率越高，则可接收分辨率的范围越大，则显示屏的适应范围越广。

我们把一个个像素点当成小格子，那么下图的分辨率就是2 \* 4的分辨率，代表横向2个像素点，纵向有4个像素点。



我们经常所说的分辨率1024\*768就是横向有1024个像素点，纵向有768个像素点，再细化一点就是输出图像的每一条水平线上包含1024个物理像素点，一共有768条水平线

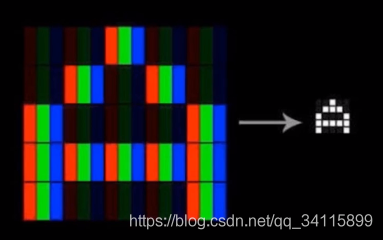
很明显，显示相同尺寸的屏幕，肯定是点越多，显示的越精细，效果越好。

那图中这些**像素点**是什么呢？其实就是**物理像素**。

2、物理像素

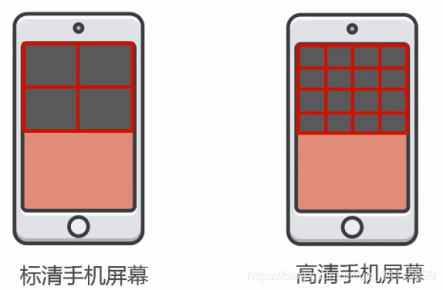
物理像素(physical pixel)又可以称为设备像素(dp : device pixel)

我们来感受一下这些物理像素点，像一个个小格子，一个个小点，看得很清楚。



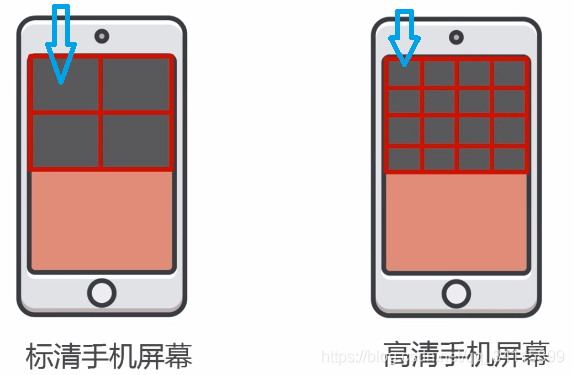
再细看每个小点，每一个小像素点都是由三原色RGB组成，显示器再控制明暗程度就可以显示想要的效果图案，从定义上来看，像素是指三原色及其灰度的基本编码。

1. 实际的开发是以物理像素为准的吗？



这两个手机的宽和高都是一样的，很显然高清屏分辨率4 \* 8比标清屏的2 \* 4要清晰一些。

假如实际开发的像素以物理像素为准，我们要显示1个像素的东西，结果如下



你看到的内容是不一样大的，很显然实际开发不是用物理像素描述的，物理像素只是为了描述设备分辨率需要知道的。实际开发其实是用CSS像素描述的。

不管是PC端还是移动端，都可以用screen.width/height来描述水平和垂直方向的物理像素，即分辨率的水平和垂直方向上的数值。注意：这里分辨率不是物理分辨率。

3、什么是物理分辨率和逻辑分辨率

手机屏幕分辨率有两个术语：物理分辨率和逻辑分辨率。那么二者的区别是什么？这里我们做下简单说明。

正如俗话所说物理分辨率是硬件所支持的，逻辑分辨率是软件可以达到的。

物理分辨率是手机显示屏实际存在的分辨率，而逻辑分辨率是我们肉眼感知的实际尺寸。

这里列举了一些机型的物理和逻辑分辨率，可以参照对比下：



（1）如何获取屏幕物理分辨率和逻辑分辨率？

前端获取逻辑分辨率的方法：

逻辑分辨率宽：window.screen.width

逻辑分辨率高：window.screen.height

前端获取物理分辨率的方法：

物理分辨率宽：逻辑分辨率宽 \* 设备像素比 （window.screen.width \* window.devicePixelRatio）

物理分辨率高：逻辑分辨率高 \* 设备像素比 （window.screen.height \* window.devicePixelRatio）

软件上认为的1px，在实际屏幕中可能有 4个 物理像素点给 这 1px 工作。

4、暂时不采用，CSS像素即设备独立像素

CSS像素可称为来**逻辑像素(logical pixel)**，也可以称为**设备独立像素**(dip : device independent pixel)

写css的时候，其实用到的就是css像素，比如

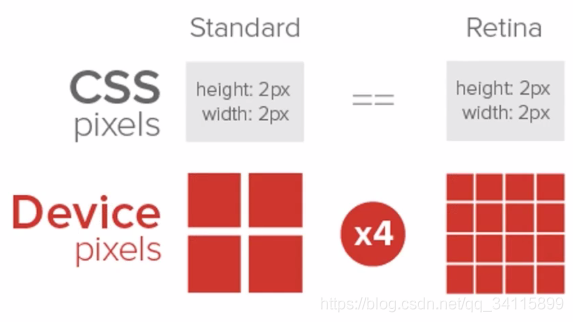
.box {

width:200px;

height:200px;

}

这里的px就是说的CSS像素，不是物理像素。那么CSS像素和物理像素是什么关系呢？来看下图



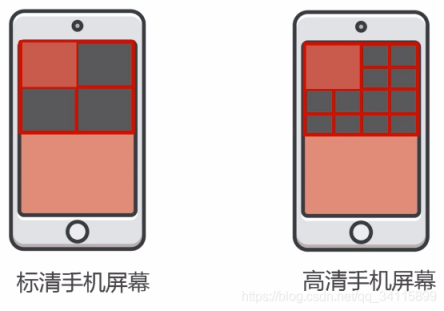
左边表示标清屏幕，右边表示视网膜高清屏幕。

宽和高都是2个CSS像素，那么在标清屏中需要用2 \* 2个物理像素来显示，即1个CSS像素用1 \* 1个物理像素来描述。

在高清屏需要4 \* 4个物理像素来显示，即1个CSS像素用2 \* 2个物理像素来描述。

高清屏就是显示每个CSS像素的物理点变多了，更精细了，看起来就更清晰了。

再来看看最初的例子



用CSS像素就可以解释上面的例子，手机实际宽和高一样，左边标清屏1个CSS像素代表1个物理像素，右边高清屏1个CSS像素代表4个物理像素。

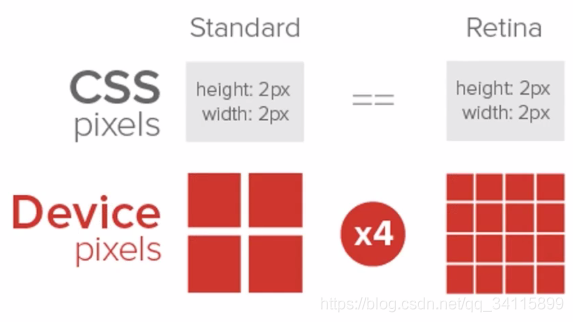
5、什么是设备像素比

设备像素比（dpr : device pixel ratio）

dpr = 同一方向上的 物理像素 / CSS像素 (缩放比是1的情况，就是css像素是可以缩放的，不要去缩放它。)

同一方向就是指的横向比或者纵向比，后面讲缩放的时候再说为什么计算dpr要求缩放比是1。

还是这张图，来讲解一下dpr



我们以一个方向为例，比如横向

标清屏：dpr = 2 / 2 = 1

高清屏(Retina)：dpr = 4 / 2 = 2

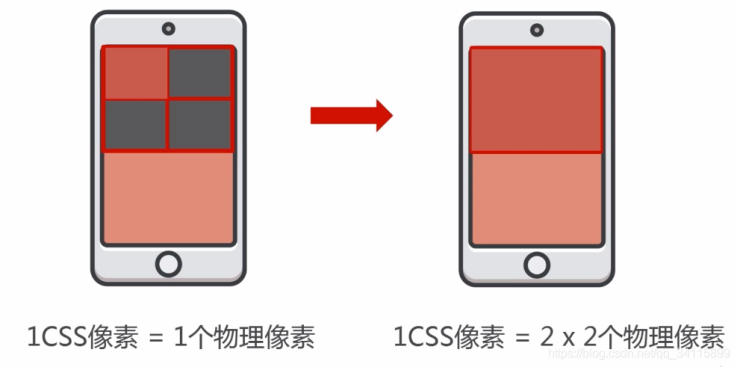
dpr = 2表示1个css像素用2 \* 2个设备像素来绘制。

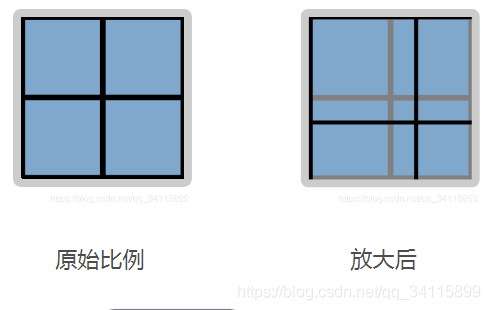
console.log(window.devicePixelRatio);可以打印dpr。

6、缩放

我们看手机经常会用手指去放大图片，看完缩小图片，缩放的是什么？**是CSS像素，不是物理像素，**物理像素是和分辨率的划分有关的，不可能通过编码来改变的。

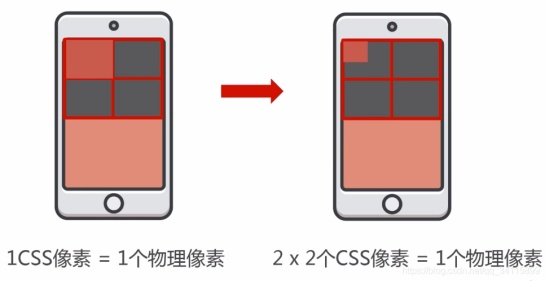
比如我们要放大，如下面2张图的示例

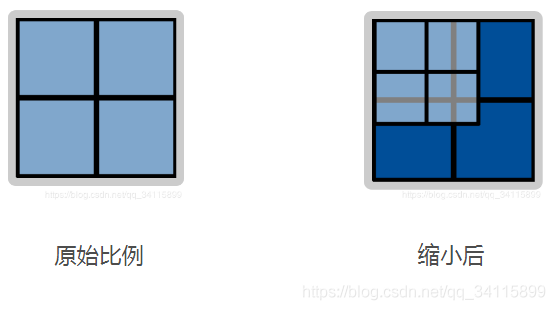




放大之后一个CSS像素与多个设备像素重叠

又比如我们要缩小，如下面2张图的示例





缩小之后一个设备像素现在重叠了几个CSS像素

缩放之后，缩放比就不为1了，所以就不能这样计算dpr了，刚刚定义dpr的时候说过了，那是缩放比必须是1才成立的定义。

7、一个300px的盒子在移动端的显示（自己初步理解）

|  |
| --- |
| 页面上的300px盒子，在1440x720（逻辑分辨率为720x360），这个300px的盒子放在手机端，它会占据300px的物理像素（例如y轴，占了1440中的300px），此时系统会根据devicePixelRatio（1440/720=2），来把这个盒子在移动端放大一下，一个css像素它会占两个物理像素，它就显示了移动端的逻辑像素300px那么大。 |

三、远程调试

1、模拟调试

现代主流浏览器均支持移动开发模拟调试，通常按F12可以调起，其使用也比较简单，可以帮我们方便快捷定位问题。机型是有限的，目前手机型号非常多

有一些机型，模拟不了，也不是很稳定，直接用真机测试。

2、真机调试

模拟调试可以满足大部分的开发调试任务，但是由于移动设备种类繁多，环境也十分复杂，模拟调试容易出现差错，所以真机调试变的非常必要。

有两种方法可以实现真机调试：

（1）将做好的网页上传至服务器或者本地搭建服务器，然后移动设备通过网络来访问。（重点），只能进行查看

（2）借助第三方的调试工具，如weinre（环境配置太复杂了）、debuggap（不稳定）、ghostlab（收费）(比较)等

真机调试必须保证移动设备同服务器间的网络是相通的。

Ghostlab使用：

|  |
| --- |
| 1. 先开启wamp，跟手机连接同个网络   2、把项目目录整个拉进去  3、通过微信扫码看页面的信息。  4、在这里能直接进行修改。 |

四、视口

视口（viewport）是用来约束网站中最顶级块元素<html>的，即**它决定了<html>的大小。**

用来显示内容的，PC端的视口大小和浏览器是一致的

1、PC设备

在PC端视口的概念并不是很重要。

在PC设备上viewport的大小取决于浏览器窗口的大小，以CSS像素做为度量单位。

通过以往CSS的知识，我们都能理解<html>的大小是会影响到我们的网页布局的，而viewport又决定了<html>的大小，所以viewport间接的决定并影响了我们网页的布局。

（1）获取viewport的大小

当你的视口在变化的时候，它里边的内容的布局是要有相对应的改变的

|  |
| --- |
| <script type="text/javascript">  console.log('视口的大小');  console.log(document.documentElement.clientWidth);  console.log(document.documentElement.clientHeight);  </script> |
| 1  2 |

（2）PC设备viewport（浏览器窗口）是如何影响我们的网页布局的

|  |
| --- |
| <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .viewport {  /\*width: 1536px;\*/  width: 100%;  height: 300px;  background-color: pink;  position: relative;  }  div {  /\*width: 384px;\*/  width: 25%;  height: 100px;  float: left;  background-color: skyblue;  position: relative;  text-align: center;  line-height: 100px;  }  div:nth-child(odd) {  background-color: red;  }  </style> |
| <ul class="viewport">  <div>1</div>  <div>2</div>  <div>3</div>  <div>4</div>  </ul> |

注：所谓正常显示是指页面布局没有错乱。

父容器宽度给设定固定值，子元素有固定值。当你的视口没有那么大的时候，而且又要你显示它里边的内容的时候，浏览器PC端默认就给你滚动条，让你可以看到。

不想要滚动条的时候，给父亲100%，但是子元素的宽度大于父元素，放不下去，所以会换行。

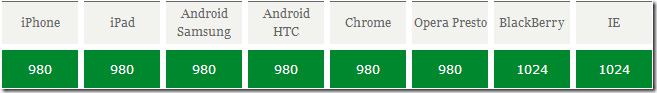
如果不想要滚动条或者换行，可以给父亲100%，子元素25%，都是百分比的情况，但是子元素会变小，有可能会造成子元素内容的溢出。造成布局错乱

（3）移动设备viewport

移动设备屏幕普遍都是比较小的，但是大部分的网站又都是为PC设备来设计的，要想让移动设备也可以正常显示网页，移动设备不得不做一些处理，通过上面的例子我们可以知道只要viewport足够大，就能保证原本为PC设备设计的网页也能在移动设备上正常显示，移动设备厂商也的确是这样来处理的。

在移动设备上**viewport不再受限于浏览器的窗口**，而是允许开发人员自由设置viewport的大小，通常浏览器会设置一个默认大小的viewport，为了能够正常显示那些专为PC设计的网页，一般这个值的大小会**大于屏幕的尺寸**。

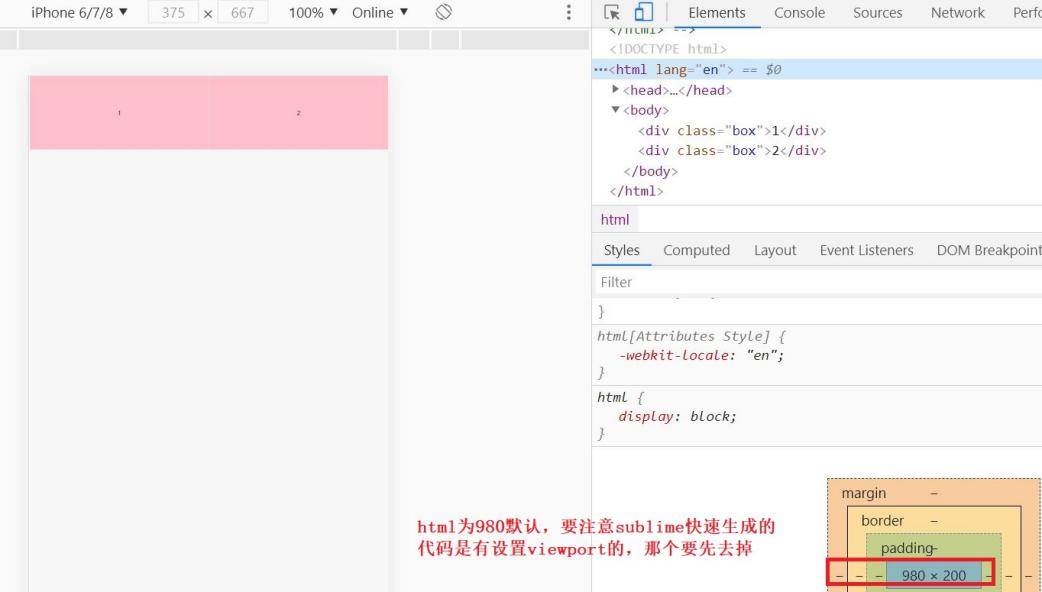
如下图为常见默认viewport大小（仅供参考）



注意：视频中的代码，是没有设置viewport的，而sublime快速生成html结构是有设置viewport，这两种结果是不一样的。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  **<meta name="viewport" content="width=device-width, initial-scale=1.0">这句话是sublime默认生成的模板中自带的，要注意删除再进行测试，因为它是设置了viewport的大小。**  <title>Document</title>  <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .box1 {  width: 490px;  height: 100px;  background-color: pink;  float: left;  }  .box2 {  width: 490px;  height: 100px;  background-color: blue;  float: left;  }  </style>  </head>  <body>  <div class="box1">1</div>  <div class="box2">2</div>  </body>  </html> |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8"> |

在F12的模拟调试中，如果页面有刷新的话，要重新进入手机的模式，可能会不准确，造成效果不一样。Android测试结果一样。



在真机中，android测试后是有出现滚动条，字有没有变小倒是看不出来。

原因：

理解两个viewport后我们来解释为什么网页会被缩放或出现水平滚动条，其原因在于移动设备浏览器会默认设置一个layout viewport，并且这个值会大于ideal viewport，那么我们也知道ideal viewport就是屏幕区域，layout viewport是我们布局网页的区域。

那么最终layout viewport是要显示在ideal viewport里的，而layout viewport大于ideal viewport时，于是就出现滚动条了。

那么为什么有的移动设备网页内容被缩放了呢？

移动设备厂商认为将**网页完整显示**给用户才最合理，而不该出现滚动条，所以就将layout viewport进行了缩放，使其恰好完整显示在ideal viewport（屏幕）里，其缩放比例为ideal viewport / layout viewport。

1. 移动设备的viewport

移动设备上有2个viewport（为了方便讲解人为定义的），分别是**layout viewport**和**ideal viewport。**

（1）layout viewport（布局视口）指的是我们可以进行网页布局区域的大小，同样是以CSS像素做为计量单位，可以通过下面方式获取

// 获取layout viewport，默认的layout viewport的大小，没有设置viewport的情况下。

document.documentElement.clientWidth;

document.documentElement.clientHeight;

如果要保证为PC设计的网页在移动设备上布局不发生错乱，移动设备会默认设置一个较大的viewport（如IOS为980px），这个 viewport 实际指的是**layout viewport**。

（2）ideal viewport（理想视口）**设备屏幕区域**，称之为物理像素，**其大小是不可能被改变**，通过下面方式可以获取。

// 获取ideal viewport有两种情形

// 新设备

window.screen.width;

window.screen.height;

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <!-- <meta name="viewport" content="width=device-width, initial-scale=1.0"> -->  <title>web 视口</title>  </head>  <body>  <div id="box"></div>  <div id="box1"></div>  <script type="text/javascript">  var box = document.getElementById('box');  var layoutWidth = document.documentElement.clientWidth;  var layoutHeight = document.documentElement.clientHeight;  box.innerText ='layout viewport宽高' + layoutWidth + ' ' + layoutHeight;  var box = document.getElementById('box1');  var layoutWidth = window.screen.width;  var layoutHeight = window.screen.height;  box.innerText ='ideal viewport宽高' + layoutWidth + ' ' + layoutHeight;  </script>  </body>  </html> |

把layout viewport的大小设置成ideal viewport的大小，这样岂不是会带来很多问题？？？

五、屏幕适配

移动页面最理想的状态是，避免滚动条且不被默认缩放处理，我们可以通过设置<meta name="viewport" content="">（这个只是在移动端会生效，name名字也是固定的）来进行控制，并改变浏览器默认的layout viewport的宽度。

1、meta设置viewport

对 viewportd 的设置的 meta 标签建议写在已有的 meta 标签之后。

content对内容进行设置

name="viewport" ：说明当前的meta标签是用来设置 viewport 的属性的，这个属性只有在移动端才会有效的，在PC端是无效的。

width：设置viewport的宽度。例如设置了300，子元素设置100%（即300px），我手机的是360。那么系统会自动将它放大，字看起来会大点；如果设置了400，那么系统会自动把它缩小，字看起来会小点。

initial-scale=1.0：设置初始化缩放比例，为 1 的话，则表示不缩放。

当我们设置width=device-width，也达到了initial-scale=1的效果，得知其实 initial-scale = ideal viewport / layout viewport（它们的结果是1，那只能说明 layout viewport的值是跟ideal viewport一样的，相当于是width = device-width）。

两种方式都可以控制缩放，开发中一般同时设置width=device-width和initial-scale=1.0（为了解决一些兼容问题）。

|  |
| --- |
| <meta name="viewport" content="**width=device-width**, **initial-scale=1.0**"> |

2、其他参数

height 设置layout viewport 高度，其取值可为数值或者device-height

一般很少会去设置的，在移动端页面库有垂直滚动条。

initital-scale设置页面的初始缩放值，为一个数字，可以带小数。

maximum-scale允许用户的最大缩放值，为一个数字，可以带小数。

minimum-scale允许用户的最小缩放值，设置默认状态下的缩放，为一个数字，可以带小数。

**要注意initial-scale也要设置小些，如果它为1，minimum-scale设置0.5，是无效的**

user-scalable是否允许用户进行缩放，值为"no"或"yes"。

注：device-width 和 device-height就是ideal viewport的宽高。

六、移动浏览器

移动端开发主要是针对IOS和Android两个操作系统平台的，除此之外还有Windows Phone。

移动端主要可以分成三大类，系统自带浏览器、应用内置浏览器、第三方浏览器

系统浏览器：指跟随移动设备操作系统一起安装的浏览器。

应用内置浏览器：通常在移动设备上都会安装一些APP例如QQ、微信（微信扫一扫就能打开页面）、微博、淘宝等，这些APP里往往会内置一个浏览器，我们称这个浏览器为应用内置浏览器（也叫WebView），这个内置的浏览器一般功能比较简单，并且客户端开发人员可以更改这个浏览器的某些设置，在我们理实的开发里这个浏览器很重要。

第三方浏览器：指安装在手机的浏览器如FireFox、Chrome、360等等。

在IOS和Android操作系统上自带浏览器、应用内置浏览器都是基于Webkit内核的。

PC端的布局有时候，是无法完全放到移动端的viewport中的，后期有两种方案：第一种就是专门做一个用于移动端展示的页面；第二种是做响应式。

移动端的相对来比较简单，受限于屏幕的大小



七、移动web京东项目首页

1. css公共样式
2. 样式重置，网页的样式是有默认值样式的，我们可能不需要，就需要重新进行初始化

（2）添加新的样式

a标签在移动端，点击的时候会出现背景。我们不需要这个背景，这是移动端特有的，pc端是看不到的，不仅仅是a标签，很多可以单击的标签都会有

解决：去除移动端特有的点击高亮效果

input标签，在获取焦点的时候，默认的边框会有阴影，outline，有时边框会有不一样，可以先去除边框，我们手动给它添加上边框。

注意点：点击logo一般可以回到主页面。

一般页面可以拉大，但是有一个最大值。不会无限拉大，一般是640px，缩小也是会达到一定宽度的。

搜索框布局思路：三栏布局，中间自适应

采用flex的话，会有兼容性问题

|  |
| --- |
| margin: 0 100px;  不能给宽度，让他默认继承父亲100% |

左右浮动也可以设置一下，清除浮动的类

字体颜色、大小在整体上基本上是差不多的

我们的logo的width、height比图片要小的时候，背景图是400x400，我们可以通过background-size来缩小背景图，缩小到我们需要的尺寸，就可以了。

heigth：100%是怎么一回事，看css

疑问：就是我把放在高分辨率情况下，图片是不是会失真

答：是会失真的



input标签的边框如果有背景颜色的话，它会有阴影什么的，这是它默认样式的影响，可以先清除一下border，在自己设置边框

问题：

发现最小宽度不起作用：多刷新几次解决。。。。要多刷新

布局好好的，超出640出现问题：是因为我把搜索框加了固定定位，fixed是相对于浏览器宽度而言的，所以要添加max-width来解决，问题就出在改的代码上，可以全部注释，通过一个个添加来定位问题，最好是实时刷新。不要一次加太多代码。

::before在input下，只用在借助其父元素，input是表单控件来着，以我浅显的理解来看，「一个元素文档树内容之前和之后的内容」就是指这个元素是要可以插入内容的，也就是说这个元素要是一个容器。input，img，iframe等元素都不能包含其他元素，所以不能通过伪元素插入内容。至于Chrome 中checkbox和radio可以插入，那应该是Bug了，要么就是姿势不对了。

background: url(images/jd-sprites.png) 0 -109px;

background-size: 200px 200px;

这个顺序是有讲究的，图片都还没有，去缩放是没有用的

1. 图片资源的话，可以分为固定图片，例如图标，一般情况下是不会变的，称之为静态图片

动态图片，例如一些产品的图片，它每次可能都不一样，需要从服务器上下载

1. 图片底边间隙的问题具体是怎么回事

3、产品模块，加了边框，不是会导致距离不够把左边盒子直接顶下来？？

答：设置了；border-sizing的问题

4、又出现了min-width无效的问题。。

答：跟user-scalable = no有关系，会影响大小，待细究

5、一般布局如果有多个类似的产品信息，可以用盒子将他们包在一起

1. 在产品块那里，为了样式格式的基本统一，但是秒杀模块又有区别，此时就可以在写类样式的时候，给他写多一个起始的类名，就是选择器不一样了，这样就不会冲突，或者直接用别的类名，也行。
2. 图片进行缩放的时候，最好是跟原图片的比例关系保持一致

8、元素基本是百分比形式出现，分成几份，不能写死了

**轮播图问题：**

1、轮播图移动端可能要考虑屏幕被缩放的问题

2、增加图片的话，要修改ul长度，跟li的占的长度

2、轮播图改比例的话，它后续的子元素的比例基本上都要改，因为它占的比例不一样了，例如在轮播图加了一张图片，然后设置ul由800变成900，li的12.5%也要进行变化(一开始是8张图，现在变成9张图了)，否则会放不下的，它的大小由原来的640变成720（640 x 900 x 12.5%）了。

4、如果需要添加一些样式，可以用添加类的方式的来做，这样也好去除

5、过渡效果，如果使用left的话，一定要将left初始化为0，否则不起作用，为什么？？？引出css资源加载跟渲染的时机。

1. \*\*js轮播图要返回第一张的时候，设置为left为0，但是发现效果还是有往回拉的效果，如果加个console.log()，能解决这儿问题，初步判断是以下的原因。

left设置0后，可能还没有渲染，接下来马上又设置了过渡，然后再设置 left的值，此时把0覆盖了，浏览器在渲染时候就直接从最后，直接拉到现在的left

js更新dom后页面不及时渲染问题（js线程阻塞和解决办法）

（1）js是同步加载的。

（2）浏览器执行js程序，高于页面渲染。

于是就出现了js更新dom，页面不能及时渲染的问题，给后面的语句加一个毫秒的延时。

No1分辨率，物理像素、视口大小，概念都混淆了

No2视口跟分辨率有关系吗

八、移动端touch事件

1. 事件类型

这些事件是在移动端才有效的。尽量使用addeventlistener

touchstart: 手指触摸屏幕时触发。只会触发一次。

touchmove: 手指在屏幕上移动时触发，前提是触摸到屏幕，也就是先有touchstart事件。会触发多次。

touchend: 手指离开屏幕时触发。只触发一次。

touchcancel触摸意外中断事件：在模拟器上可能无法模拟，例如在输入文字时候，有电话进来，此时触摸会被中断。

|  |
| --- |
| <div class="nav"></div>  <script type="text/javascript">  var nav = document.querySelector('.nav');  nav.addEventListener('touchstart', function () {  console.log('touchstart');  });  nav.addEventListener('touchmove', function () {  console.log('touchmove');  });  nav.addEventListener('touchend', function () {  console.log('touchend');  });  </script> |

2、TouchEvent对象

touches: 是指当前屏幕上所有的手指对象，你按下来就有记录。

targetTouches: 位于该元素上的所有手指的列表

changedTouches：当前屏幕上变换的手指对象，从有到无，从无到有。（好像一次就一个）

在touchstart事件中打印这个对象，当第一个手指按下的时候，从无到有，它就记下了，它里边有个id值都不一样。

在touchend事件中，当手指离开的时候，他也会打印，这个id跟按下的时候那个id是一致的。

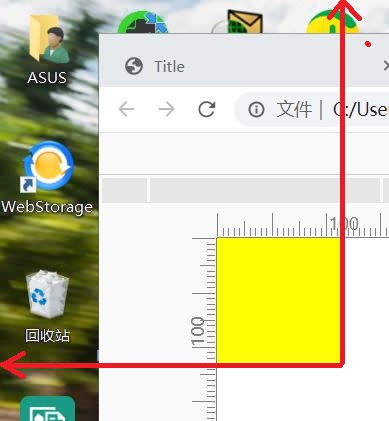


3、Touch对象

（1）clientX / Y：相对于当前视口，移动端屏幕。一般使用的是这，因为对视口设置了大小，content="width=device-width"。pageX / Y一般是一样的

（2）pageX / Y：相对于当前页面的内容，会有滚动条，包含滚动条的距离。触点相对于HTML文档左边沿的的X坐标. 和 clientX 属性不同, 这个值是相对于整个html文档的坐标, 和用户滚动位置无关. 因此当存在水平滚动的偏移时, 这个值包含了水平滚动的偏移.

（3）screenX / Y：当前的触摸点，距离你屏幕左上角（PC端）的坐标距离，在chrome 模拟器获取可能不是很准确。



|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" **content="width=device-width**, initial-scale=1.0, user-scalable=no">  <title>CSS测试</title>  <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .nav {  text-align: center;  **width: 800px; // 超过才有滚动条**  height: 200px;  background: linear-gradient(to right ,yellow, skyblue);  }  </style>  </head>  <body>  <div class="nav"></div>  <script type="text/javascript">  var nav = document.querySelector('.nav');  nav.addEventListener('touchstart', function (e) {  console.dir(e);  var clientX = Math.floor(e.targetTouches[0].clientX);  var clientY = Math.floor(e.targetTouches[0].clientY);  var pageX = Math.floor(e.targetTouches[0].pageX);  var pageY = Math.floor(e.targetTouches[0].pageY);  var str = 'client:'+clientX +','+ clientY+'\n'+'page:'+pageX +','+pageY;  this.innerText = str;  });  </script>  </body>  </html>  1 2 |

4、拖拽效果

视频中使用translate方法只能用一次，不太实用。

目前还是采用定位方式来搞

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">  <title>CSS测试</title>  <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .nav {  text-align: center;  width: 100px;  height: 100px;  background: linear-gradient(to right ,yellow, skyblue);  position: relative;  }  </style>  </head>  <body>  <div class="nav"></div>    <script type="text/javascript">  var nav = document.querySelector('.nav');  var startX = 0, startY = 0;  var distanceX = 0, distanceY = 0;  nav.addEventListener('touchstart', function (e) {  startX = Math.floor(e.targetTouches[0].clientX);  startY = Math.floor(e.targetTouches[0].clientY);  console.log(startX, startY);  });  nav.addEventListener('touchmove', function (e) {  var x = Math.floor(e.targetTouches[0].clientX);  var y = Math.floor(e.targetTouches[0].clientY);  distanceX = x - startX;  distanceY = y - startY;  console.log(distanceX, distanceY);  nav.style.transform = 'translate('+ distanceX +'px, '+ distanceY +'px)';  });  </script>  </body>  </html> |
| 定位的方式  <script type="text/javascript">  var nav = document.querySelector('.nav');  var startX = 0, startY = 0;  nav.addEventListener('touchstart', function (e) {  startX = Math.floor(e.targetTouches[0].clientX) - nav.offsetLeft;  startY = Math.floor(e.targetTouches[0].clientY) - nav.offsetTop;  console.log(startX, startY);  });  nav.addEventListener('touchmove', function (e) {  var left = Math.floor(e.targetTouches[0].clientX) - startX;  var top = Math.floor(e.targetTouches[0].clientY) - startY;  nav.style.left = left + 'px';  nav.style.top = top + 'px';  });  </script> |

1. e.target的使用

利用事件委托，可以让多个元素能直接拖拽。

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">  <title>CSS测试</title>  <style type="text/css">  \* {  margin: 0;  padding: 0;  }  .nav {  text-align: center;  width: 100px;  height: 100px;  background: linear-gradient(to right ,yellow, skyblue);  position: absolute;  }  .nav:nth-child(2) {  top: 100px;  }  </style>  </head>  <body>  <div class="nav">1</div>  <div class="nav">2</div>  <script type="text/javascript">  var nav = document.querySelector('.nav');  var startX = 0, startY = 0;  **document**.addEventListener('touchstart', function (e) {  startX = Math.floor(e.targetTouches[0].clientX) - e.target.offsetLeft;  startY = Math.floor(e.targetTouches[0].clientY) - e.target.offsetTop;  console.log(startX, startY);  });  **document**.addEventListener('touchmove', function (e) {  var left = Math.floor(e.targetTouches[0].clientX) - startX;  var top = Math.floor(e.targetTouches[0].clientY) - startY;  e.target.style.left = left + 'px';  e.target.style.top = top + 'px';  });  </script>  </body>  </html> |

我们以mac的发展史来说明什么是物理分辨率，以及为什么需要衍生出逻辑分辨率。

我们的主人公是乔帮主和比尔盖茨。此时乔帮主面前有一台mac，屏幕的分辨率是1280\*720，这就是物理分辨率。乔帮主对比尔盖茨说，给我的mac开发一个word软件吧。盖茨说OK，于是写了一个软件，这个软件显示的时候长度是1280像素，宽度是720像素，正好能够盖满整个mac屏幕。乔帮主看了之后很满意。

有一天，乔帮主看自己的mac屏幕觉得很粗糙，一点都不清晰锐利。于是聪明的乔帮主想到，同样是15寸的屏幕，我把像素点翻倍，不就可以更清晰了吗？于是他把mac的物理分辨率改成了2560\*1440的分辨率，相当于每个像素点的尺寸减少了4倍，这下再也看不出颗粒感了，乔帮主很满意。

但是，当乔帮主打开盖茨给他写的word的时候，傻眼了，原本全屏的word现在只占屏幕的四分之一，而且文字非常的小。乔帮主打电话给盖茨说，你的软件怎么出问题了？盖茨回答说，我开发的时候你的mac分辨率就是1280\*720，你自己改了硬件尺寸怪我咯，我很忙没空给你改软件代码，你就凑合着用吧。

乔帮主稍作思考，马上想出了一个非常聪明的主意！他在软件和硬件之间的系统层加了一层逻辑分辨率。虽然屏幕横向有2560个像素点，但是告诉软件我只有1280个像素点！当word的宽度要占1280个像素的时候，实际上已经占了2560个像素。于是我们的word又占满屏幕了！

于是乔帮主把这个机智的想法命名为逻辑分辨率，不管我显示器的硬件有多少个像素点，我只会告诉软件我的逻辑分辨率，这样软件的代码就不用修改也能在不同的屏幕上显示效果一致。

在多年后，imac的物理分辨率已经达到了5120 x 2880，但是告诉软件的时候还是说我的分辨率是1280\*720，相当在盖茨看来的一个逻辑像素点，背后实际上已经有16（4x4）个物理像素点为其工作了！

网页上的1px跟手机端的1px是不一样大小的，因为它们的工艺是不一样的