

本文介绍了移动端适配的 3 种方法，以及移动端图片模糊问题和 1px 细线问题的解决方法。当然了，在这之前先整理了与这些方法相关的知识：物理像素、设备独立像素、设备像素比和 viewport。

>>>>物理像素、设备独立像素和设备像素比

在 CSS 中我们一般使用 px 作为单位，需要注意的是，CSS 样式里面的 px 和物理像素并不是相等的。CSS 中的像素只是一个抽象的单位，在不同的设备或不同的环境中，CSS 中的 1px 所代表的物理像素是不同的。在 PC 端，CSS 的 1px 一般对应着电脑屏幕的 1 个物理像素，但在移动端，CSS 的 1px 等于几个物理像素是和屏幕像素密度有关的。

物理像素(physical pixel)

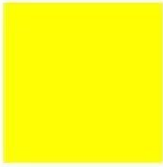
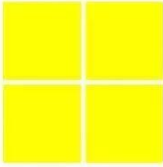
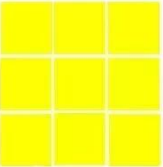
物理像素又被称为设备像素、设备物理像素，它是显示器（电脑、手机屏幕）最小的物理显示单位，每个物理像素由颜色值和亮度值组成。所谓的一倍屏、二倍屏(Retina)、三倍屏，指的是设备以多少物理像素来显示一个 CSS 像素，也就是说，多倍屏以更多更精细的物理像素点来显示一个 CSS 像素点，在普通屏幕下 1 个 CSS 像素对应 1 个物理像素，而在 Retina 屏幕下，1 个 CSS 像素对应的却是 4 个物理像素（参照下文田字示意图理解）。

设备独立像素(device-independent pixel)

设备独立像素又被称为 CSS 像素，是我们写 CSS 时所用的像素，它是一个抽象的单位，主要使用在浏览器上，用来精确度量 Web 页面上的内容。

设备像素比(device pixel ratio)

设备像素比简称为 dpr，定义了物理像素和设备独立像素的对应关系：设备像素比 = 物理像素 / 设备独立像素。

	普通屏	二倍屏	三倍屏
设备独立像素	weight : 1px height : 1px	weight : 1px height : 1px	weight : 1px height : 1px
物理像素			

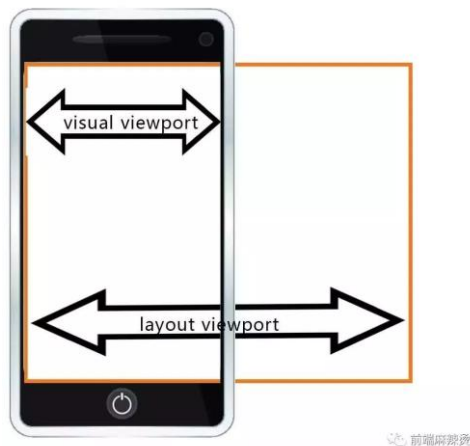
前端麻辣烫

CSS 的 1px 等于几个物理像素，除了和屏幕像素密度 dpr 有关，还和用户缩放有关系。例如，当用户把页面放大一倍，那么 CSS 中 1px 所代表的物理像素也会增加一倍；反之把页面缩小一倍，CSS 中 1px 所代表的物理像素也会减少一倍。关于这点，在文章后面的 1px 细线问题部分还会讲到。

>>>>viewport

viewport 就是设备上用来显示网页的那一块区域，但 viewport 又不局限于浏览器可视区域的大小，它可能比浏览器的可视区域要大，也可能比浏览器的可视区域要小。在默认情况下，一般来讲，移动设备上的 viewport 都是要大于

浏览器可视区域的，这是因为考虑到移动设备的分辨率相对于桌面电脑来说都比较小，所以为了能在移动设备上正常显示那些传统的为桌面浏览器设计的网站，移动设备上的浏览器都会把自己默认的 viewport 设为 980px 或 1024px（也可能是其它值，这个是由设备自己决定的），但带来的后果就是浏览器会出现横向滚动条，因为浏览器可视区域的宽度是比这个默认的 viewport 的宽度要小的。



明确三种不同的 viewport 视口：

visual viewport 可见视口，指屏幕宽度

layout viewport 布局视口，指 DOM 宽度

ideal viewport 理想适口，使布局视口就是可见视口即为理想适口

获取屏幕宽度(visual viewport)的尺寸：

```
window. innerWidth/Height
```

获取 DOM 宽度(layout viewport)的尺寸：

```
document. documentElement. clientWidth/Height
```

设置理想视口 ideal viewport：

```
<meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
```

该 meta 标签的作用是让 layout viewport 的宽度等于 visual viewport 的宽度，同时不允许用户手动缩放，从而达到理想视口。

meta[name="viewport"]里各参数的含义为：

width: 设置 layout viewport 的宽度，为一个正整数，或字符串“width=device”。

initial-scale: 设置页面的初始缩放值，为一个数字，可以带小数。

minimum-scale: 允许用户的最小缩放值，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。

height: 设置 layout viewport 的高度，这个属性对我们并不重要，很少使用。

user-scalable: 是否允许用户进行缩放，值为 “no” 或 “yes” 。

>>>>rem 适配方案

适配是为了使页面在不同手机设备上，相对保持统一的效果。移动端自适应方案很多，有百分比布局，弹性盒模型布局等，但是最好用的要数 rem 布局了。

rem 是相对于根元素的字体大小的单位，我们可以根据设备宽度动态设置根元素的 font-size，使得以 rem 为单位的元素在不同终端上以相对一致的视觉效果呈现。下面介绍 3 种根据屏幕宽度设置 rem 基准值的方法。（注：为了换算方便，以下三种方法都用 1:100 的比例，即 1rem=100px。）

用 JS 设置 rem 基准值

```
/* 设计稿是 750,采用 1 : 100 的比例,用 1rem 表示 100px,font-size 为 100 * (clientWidth / 750) */
(function(doc, win) {
    var docEl = doc.documentElement,
        resizeEvt = 'orientationchange' in window ? 'orientationchange' : 'resize',
        recalc = function() {
            var clientWidth = docEl.clientWidth;
            if (!clientWidth) return;
            docEl.style.fontSize = 100 * (clientWidth / 750) + 'px';
        };
    if (!doc.addEventListener) return;
    win.addEventListener(resizeEvt, recalc, false);
    doc.addEventListener('DOMContentLoaded', recalc, false);
})(document, window);
```

用密集的媒体查询设置 font-size

```
/* 设计稿是 750,采用 1 : 100 的比例,用 1rem 表示 100px,100*(100/750)=13.333 以 min-width: 750px 时 font-size: 100px 为基准,
min-width 每缩小 100px,font-size 就缩小 13.3333px, 如需更密集的媒体查询可以按照这个对照关系设置。*/
@media screen and (min-width: 320px) {
    html {
        font-size: 42.6667px;
    }
}
```

```

}
@media screen and (min-width: 375px) {
  html {
    font-size: 50px;
  }
}
@media screen and (min-width: 425px) {
  html {
    font-size: 56.6667px;
  }
}
@media screen and (min-width: 768px) {
  html {
    font-size: 102.4px;
  }
}

```

用单位 vw 设置 font-size

1vw 等于屏幕可视区宽度(的可视区域的百分之一)。

```

/* 设计稿是 750,采用 1 : 100 的比例,用 1rem 表示 100px,font-size 为 100*(100vw/750) */
html {
  font-size: 13.3334vw;
}

```

注:兼容性不是很好。

了解了物理像素、设备独立像素、设备像素比和 viewport 这几个重要概念后,来看一下移动端开发中,由于屏幕分辨率导致的两个经典问题:图片模糊问题和 1px 细线问题。(注:为了叙述简洁,以下多倍屏均只叙述 2 倍 Retina 屏,其它屏幕同理。)

>>>> 图片模糊问题

一个位图像素是栅格图像(如 png, jpg, gif 等)最小的数据单元。每一个位图像素都包含着一些自身的显示信息(如:显示位置,颜色值,透明度等)。理论上,1 个位图像素对应于 1 个物理像素,图片才能得到完美清晰的展示。对于 dpr=2 的 Retina 屏幕而言,1 个位图像素对应于 4 个物理像素,由于单个位图像素不可以再进一步分割,所以只能就近取色,导致图片看起来比较模糊,如下图。



对于图片模糊问题,比较好的方案就是用多倍图片(@2x)。如:一个 200×300(CSS pixel)的 img 标签,对于 dpr=2 的屏幕,用 400×600 的图片,如此一来,位图像素点个数就是原来的 4 倍,在 Retina 屏幕下,位图像素点个数就可以跟物理像素点个数形成 1:1 的比例,图片自然就清晰了。

如果普通屏幕下,也用了两倍图片,会怎样呢?

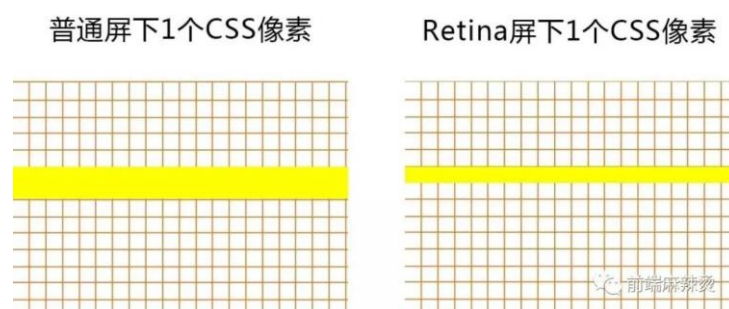
在普通屏幕下,200×300(CSS pixel)img 标签,所对应的物理像素个数就是 200×300 个,而两倍图片的位图像素个数是 200×300×4 个,所以就出现一个物理像素点对应 4 个位图像素点,但它的取色也只能通过一定的算法取某一个位图像素点上的色值,这个过程叫做(downsampling),肉眼看上去虽然图片不会模糊,但是会觉得图片缺少一些锐利度,或者是有点色差,如下图。



所以最好的解决办法是:不同的 dpr 下,加载不同的尺寸的图片。不管是通过 CSS 媒体查询,还是通过 JS 条件判断都是可以的。

>>>> 1px 细线问题

在上文我们已经知道,CSS 像素为 1px 宽的直线,对应的物理像素是不同的,可能是 2px 或者 3px,而设计师想要的 1px 宽的直线,其实就是 1 物理像素宽,如下图。



对于 CSS 而言,可以认为是 border: 0.5px; 这是多倍屏下能显示的最小单位。然而,并不是所有手机浏览器都能识别 border: 0.5px, 有的系统里,0.5px 会被当成为 0px 处理,那么如何实现这 0.5px 呢? 网上有很多解决方法,比如 border-image 图片、background-image 渐变、box-shadow 等,因为这些方案不太好,所以不做赘述了,我推荐两种方法:用媒体查询根据 dpr 用“伪元素+transform”对边框进行缩放;用 JS 根据屏幕尺寸和 dpr 精确地设置不同屏幕所应有的 rem 基准值和 initial-scale 缩放值。

伪元素+transform

构建 1 个伪元素, border 为 1px, 再以 transform 缩放到 50%。

```

/* 设计稿是 750,采用 1 : 100 的比例,font-size 为 100*(100vw/750) */
.border-1px {
    position: relative;
}
@media screen and (-webkit-min-device-pixel-ratio: 2) {
    .border-1px:before {
        content: " ";
        position: absolute;
        left: 0;
        top: 0;
        width: 100%;
        height: 1px;
        border-top: 1px solid #D9D9D9;
        color: #D9D9D9;
        -webkit-transform-origin: 0 0;
        transform-origin: 0 0;
        -webkit-transform: scaleY(0.5);
        transform: scaleY(0.5);
    }
}

```

用 JS 计算 rem 基准值和 viewport 缩放值

```

/* 设计稿是 750,采用 1 : 100 的比例,font-size 为 100 * (docEl.clientWidth * dpr / 750) */
var dpr, rem, scale;
var docEl = document.documentElement;
var fontEl = document.createElement('style');
var metaEl = document.querySelector('meta[name="viewport"]');
dpr = window.devicePixelRatio || 1;
rem = 100 * (docEl.clientWidth * dpr / 750);
scale = 1 / dpr;
// 设置 viewport , 进行缩放 , 达到高清效果
metaEl.setAttribute('content', 'width=' + dpr * docEl.clientWidth + ',initial-scale=' + scale +
',maximum-scale=' + scale + ', minimum-scale=' + scale + ',user-scalable=no');
// 设置 data-dpr 属性 , 留作的 css hack 之用 , 解决图片模糊问题和 1px 细线问题
docEl.setAttribute('data-dpr', dpr);
// 动态写入样式
docEl.firstElementChild.appendChild(fontEl);
fontEl.innerHTML = 'html{font-size:' + rem + 'px!important;}';

```

相较于上文 rem 适配方案里“用 JS 计算 rem 基准值”的方案，这个“用 JS 计算 rem 基准值和 viewport 缩放值”的方案可以解决 1px 细线问题。表格以 2 倍 Retina 屏做比较，其他多倍屏同理。

	用 JS 计算 rem 基准值	用 JS 计算 rem 基准值和 viewport 缩放值
CSS 像素为 750px 的普通屏	1rem =100px initial-scale=1 1 个 CSS 像素=1 个物理像素	1rem =100px initial-scale=1 1 个 CSS 像素=1 个物理像素
CSS 像素为 750px 的 Retina 屏	1rem =100px initial-scale=1 1 个 CSS 像素=2 个物理像素	1rem =200px initial-scale=0.5 1 个 CSS 像素=1 个物理像素

用 JS 根据屏幕尺寸和 dpr 精确地设置不同屏幕所应有的 rem 基准值和 initial-scale 缩放值 这个 JS 方案已经在完美解决了 1px 细线问题，我们不需要再做任何事情，至于图片模糊问题，只需要根据 data-dpr 的值动态加载不同尺寸的图就可以了。