# 46 响应式布局原理与实践（下）

更新时间：2020-06-16 09:44:16

> 受苦的人，没有悲观的权利。——尼采

在上一节，我们已经理解了响应式布局中几个最容易答错的概念。本节，我们来学习响应式布局相关的解决方案。

## 什么是响应式布局

响应式布局的目的是为了让我们的页面能够在不同大小的设备屏幕上正常展示。

响应式概念的提出，和移动互联网的飞速发展有着密不可分的关系。如今，同一张页面既可能展示在 PC 端，也可能展示在不同型号的 iPad、手机等各种五花八门的移动电子设备上。能够使一张页面适配多种屏幕的布局方案，就是所谓的“响应式布局方案”。

具体来说，**响应式布局主要解决的是屏幕大小不确定的问题。**

## 响应式布局方案面面观

关于响应式布局，从古到今也涌现过不少有趣的办法。在本节，我希望大家能够着重掌握的有以下三种：

- 媒体查询
- rem
- vw/vh

# 媒体查询

是什么

既然要解决的是屏幕大小不确定的问题，那么最直接的思路就是想办法去感知屏幕大小的变化，并根据不同的屏幕大小展示不同的样式。媒体查询做的就是这件事情，它是一个古老而经典的响应式布局解决方案，是 BootStrap 响应式特性的基石。

怎么用

媒体查询对应到编码上非常好理解，它是一段形如这样的 css：

```
@media screen and (max-width: 320px) {
    div {
     width: 160px;
    }
}

@media screen and (min-width: 768px) {
    div {
     width: 300px;
    }
}
```

其中，@media 是媒体查询属性的标识，“screen”指的是媒体类型。最关键的是跟在 and 后面的逻辑操作符，这段 demo 中给出的 min-width 和 max-width 更是重点中的重点：

- max-width：对最大宽度的限制。比如我们第一条媒体查询声明语句中，max-width: 320px，它的意思就是说当设备屏幕宽度不大于320px时，则采纳这条声明对应的样式规则。
- min-width：对最小宽度的限制。比如我们第一条媒体查询声明语句中，min-width: 768px，它的意思就是说当设备屏幕宽度不小于768px时，则采纳这条声明对应的样式规则。

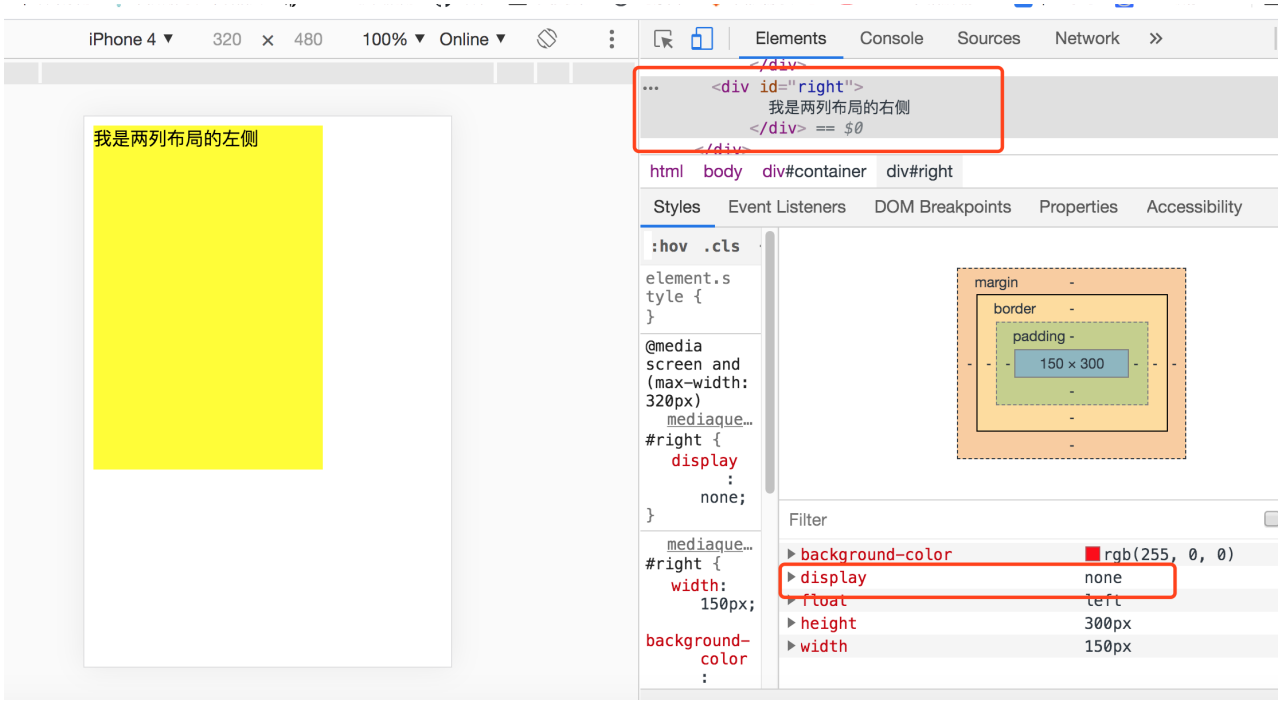
示例

下面我们来看一个通过媒体查询实现展示内容自适应的例子：

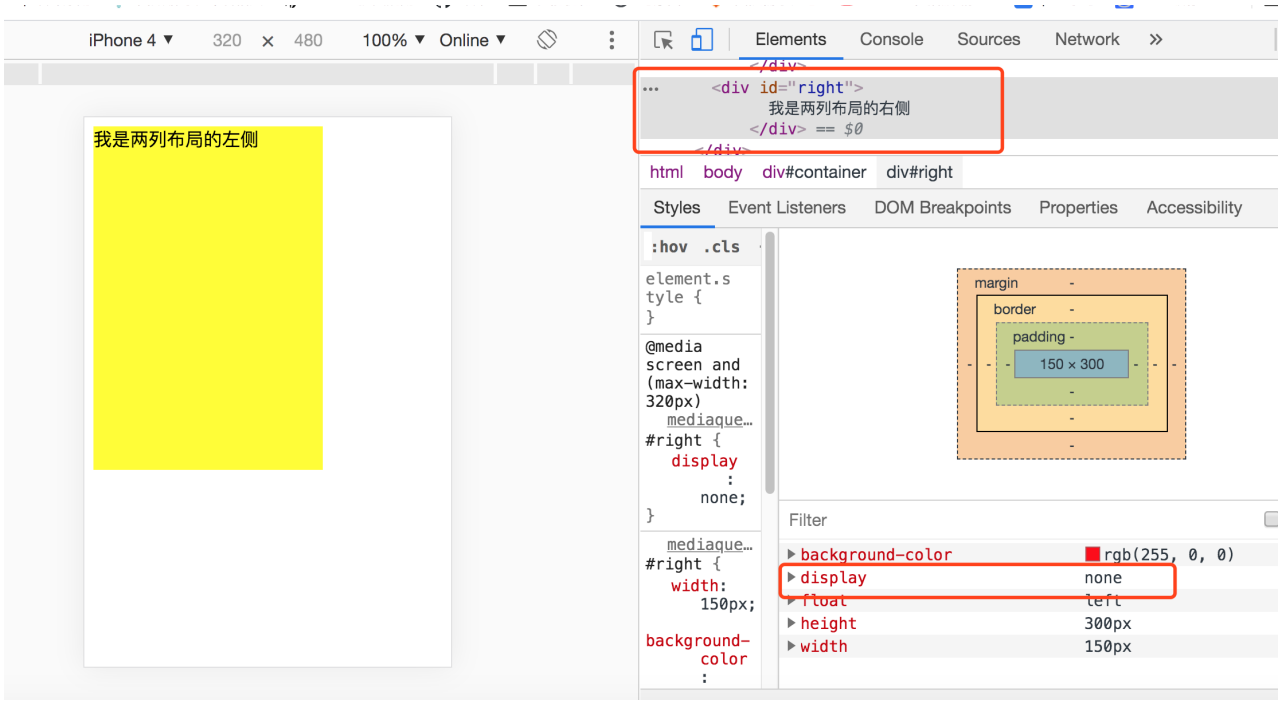
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>媒体查询示例</title>
  <style>
    #left,
    #right {
      height: 300px;
      float: left;
    }
    #right {
      width: 150px;
      background-color: red;
    }
    #left {
      width: 200px;
      background-color: yellow;
    }
    @media screen and (max-width: 320px) {
    #right {
    display: none;
    }
    }

    @media screen and (min-width: 768px) {
    #right {
    width: 300px;
    }
    }
  </style>
</head>
<body>
  <div id="container">
    <div id="left">
      我是两列布局的左侧
    </div>
    <div id="right">
      我是两列布局的右侧
    </div>
  </div>
</body>
</html>
```

以上代码通过应用浮动布局，实现了一个大家都非常熟悉的两列边栏。

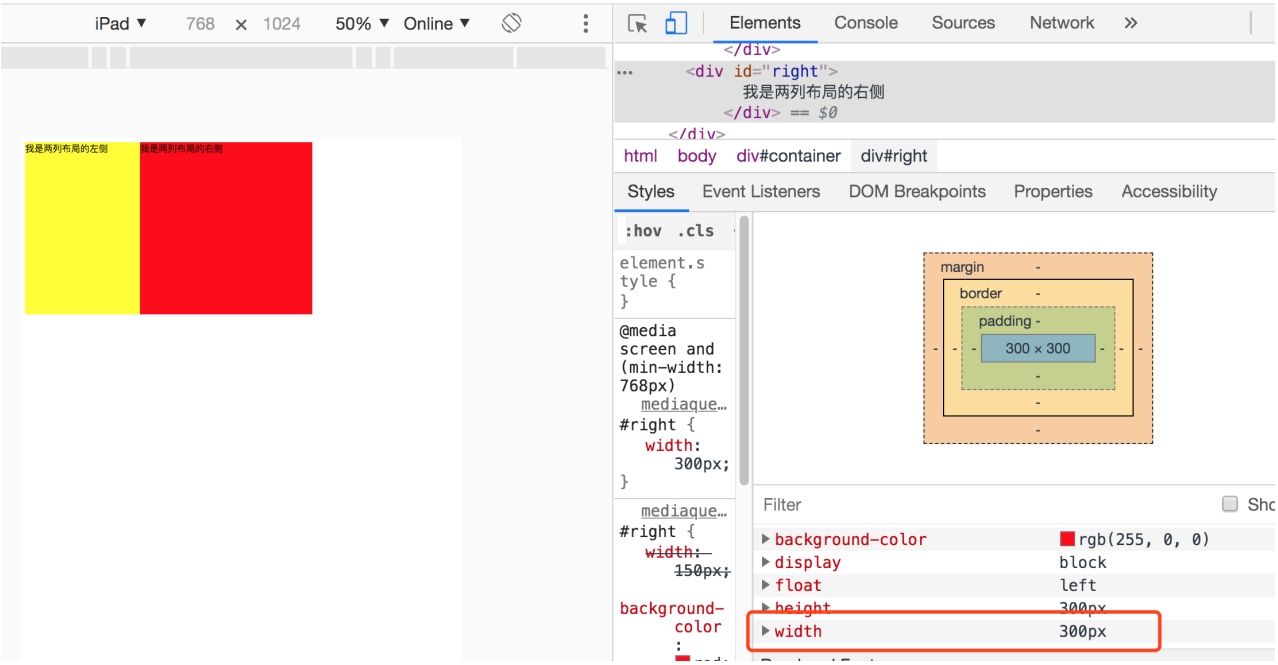
这个页面在较小的屏幕下（宽度不大于320px，一般来说是　iphone5及以下版本），考虑到空间有限，对右侧边栏进行了隐藏处理：




而在稍微大一点的屏幕下（介于320px和768px之间，一般来说仍然在手机屏幕的尺寸范围内），宽度增长到了150px：




在更大的屏幕下（不小于768px，一般来说是 iPad 或者 PC 屏幕），宽度则增长到了比左侧边栏更宽的300px：




总结

在这个例子中，页面根据不同的宽度节点，能够展示出三种不同的元素样式。这也正是媒体查询的魅力所在。

通过使用媒体查询，我们可以灵活地控制页面中的元素样式，使其能够“因地制宜”、在不同大小的设备上有不同的表现，进而确保整体页面展示效果的合理性。

## rem

上一节我们已经知道，rem 是一个以根元素 font-size 大小为基准的相对单位。如果我们以 rem 作为布局单位，那么只要根元素大小发生了改变，就有“牵一发而动全身”的效果，整个页面中所有相关元素的大小都会跟着进行相应的放缩。如果我们能够根据设备屏幕大小的不同，动态地修改根元素的 font-size，那么就相当于间接地修改了页面中所有元素的大小，进而实现了响应式布局。

这个思路用编码实现如下：

```
function refreshRem() {
  // 获取文档对象(根元素)
  const docEl = doc.documentElement;
  // 获取视图容器宽度
  const docWidth = docEl.getBoundingClientRect().width;
  // rem 取值为视图容器宽度的十分之一大小
  const rem = width / 10;
  // 设置 rem 大小
  docEl.style.fontSize = rem + 'px';
}
// 监听浏览器窗口大小的变化
window.addEventListener('resize', refreshRem);
```

以上这段代码节选自一个非常经典的轮子——flexible.js（flexible.js 是手淘前端团队整合的一套相当成熟的移动端自适应解决方案库，这里推荐学有余力的同学私下可以多做一些了解）。

在这个示例中，我们将 rem 固定为视图容器宽度的十分之一。之后不管视图宽度如何变化，1rem 始终都是视图宽度的 1/10。此时使用 rem 来进行布局，就可以实现等比缩放。

# vw/vh

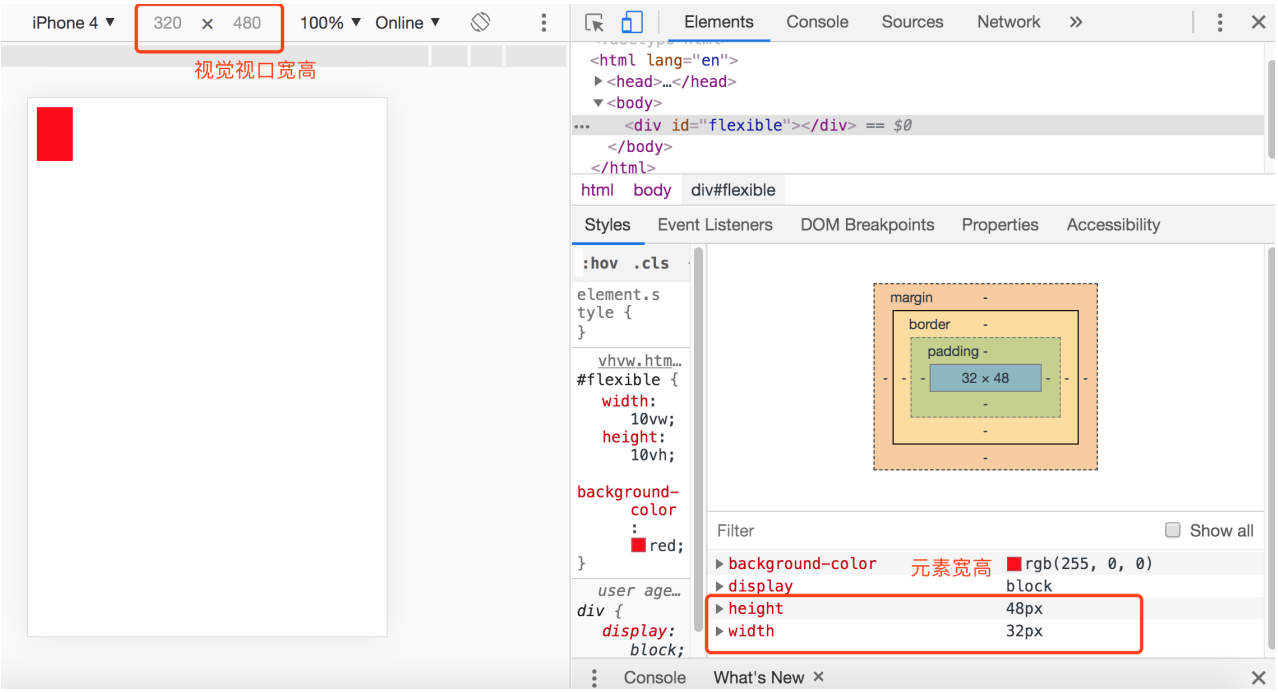
vw 和 vh 是一种区别与 rem 和 px 的 css 尺寸单位。它们天生自带等比缩放能力：

- vw：1vw = 视觉视口宽度 / 100
- vh：1vh = 视觉视口高度 / 100

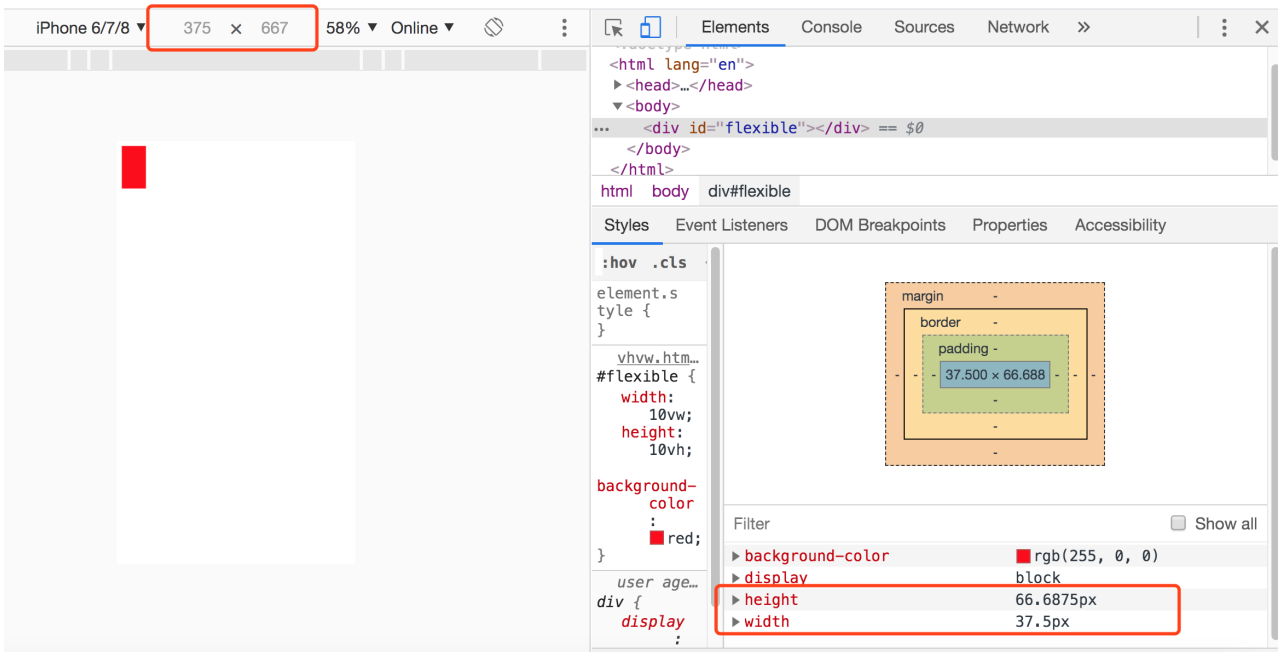
我们用一个例子来理解一下两个概念：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vw/vh实例</title>
  <style>
    #flexible {
      width: 10vw;
      height: 10vh;
      background-color: red;
    }
  </style>
</head>
<body>
  <div id="flexible"></div>
</body>
</html>
```

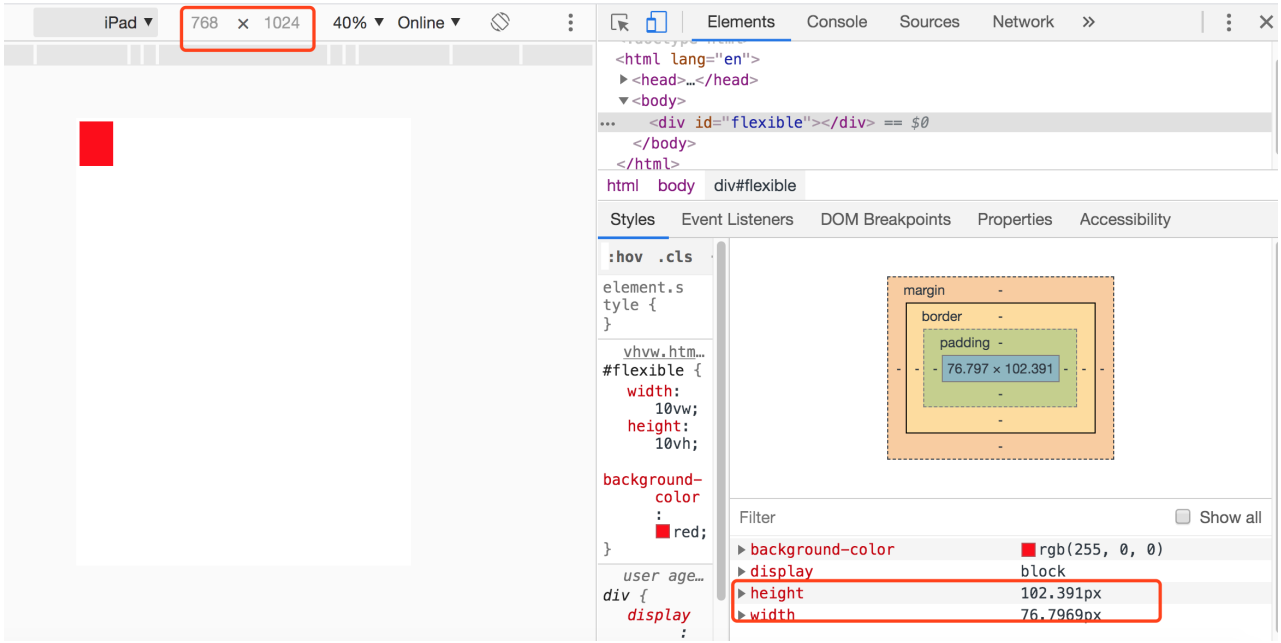
这里我设置了一个 div 元素，它的宽高分别为 10vw 和 10vh。按照刚刚讲过的换算关系，这个元素应该始终占据整个页面宽度/高度的十分之一。我们先来看看它在 iphone4 下的表现：




稍微把设备屏幕调大一点，看一下 iphone6/7/8 下的表现：




最后用 iPad 来试一下：




在以上三个示例中，不管窗口大小如何变化，div 元素的宽高都稳稳地维持在整体宽高的 1/10 水平。

相信不少同学此时已经有了这种感觉：vw 和 vh 在自适应能力上，和 refreshRem+rem 的方案效果是一样的。从实现上来说，比后者简单不少。

那么是不是意味着 vw/vh 就是响应式布局的最佳解决方案呢

答案是否定的。对于本文所提及的三种响应式解决方案，大家要清楚两点：

1. 这三种方案之间不是互斥的关系。在实践中，我们会经常遇到 rem+媒体查询、vw/vh+媒体查询这样的应用场景。
2. rem 和 vw/vh 之间不存在绝对的优劣，在选型上需要注意的是兼容性：vw/vh 的兼容性不如 rem。早年 rem 一直是响应式布局的主流解决方案，最关键的原因就是兼容性好，不挑食。而 vw/vh 则需要ios8、安卓4.4及以上操作系统的支持。话虽如此，2020年了，低于 ios8 和安卓 4.4 的机型在市场中的占比并不高（不过大家选型

的时候还是要根据自己产品的投放情况实事求是地做判断哈），rem 和 vw/vh 的选择在如今更像是一个仁者见仁智者见智的问题。

## 高清方案

说到这里，不知道大家还记不记得我们在本章第一节留下的那个悬念：用 viewport 缩放来解决 1px 问题后留下的副作用，如何解决？

这个问题有一个系统化的解决办法——高清方案。高清方案是移动端适配中相对高阶的知识，对综合能力要求较高。因此接下来的讲解，要求同学对前面三节的内容有非常深刻的理解和掌握。初次学习这块知识的同学，可能会觉得有点“绕”，是非常正常的——“绕”是知识结构不稳定导致的。如果你有类似的感觉，也不要灰心，可以尝试将本章所涉及的各项知识点整合起来，逐个映射到下面的讲解内容中去，这样做会很大程度上辅助到你的理解。

首先，一起来回顾一下这个副作用的内容：

> 1px 的 bug 就这样轻松搞定了，但这样做的副作用也很大，整个页面被缩放了。这时候我们的 1px 已经被处理成物理像素大小了，这样的大小在手机上显示边框很合适。但仔细想想，一些原本不需要被缩小的内容，比如文字、图片等，也被无差别缩小掉了。

没错，页面上的图片和文字都被缩小了 dpr 倍，现在我们需要把原来写的 css 像素都再乘以 dpr 倍。这样才能既解决 1px 问题，又与原来大小保持一致。

再来看公式：**viewport-width = screen.width * dpr ， dpr = 设备物理像素 / screen.width**， 因此设置**viewport = 设备物理像素** 即可达到目的，如果视觉稿一开始就以设备物理像素来给出，那么我们就不用再乘以 dpr 倍了，直接设计稿里 px 多少，css 里 px 就写多少即可。

那如果设计稿和设备物理像素不一样怎么办？

比如设计稿是 750 的，手机 **screen.width = 375 dpr = 3**，按照上面的公式，**viewport-width = 1125**，那么按照设计稿来写 px，虽然高清了但肯定是铺不满的，还需要再适配一下。

**这个适配问题就可以用 rem 来解决**。我们把视觉稿总的 fontSize 记为 baseFontSize（我们团队设置的是100），然后将 baseFontSize 记为rem，这样一来 1px=0.01rem(因为 1rem = 100px)，最后针对不同手机屏幕尺寸和dpr动态改变根节点html的 font-size 大小即可。

```
rem = _baseFontSize / _psdWidth * docEl.clientWidth;
```

在这个公式里，假如视觉规范是 750（_psdWidth = 750），那么我们就确保了 rem 无论在任何页面里，都一定是占据了这个页面的 100/750 这个比例，也就是说 1px = 0.01rem 永远占据这个页面的 1/750，如此便能够实现不同宽度机型的适配啦。

读到这里，大家会发现 rem 相对于 vw/vh 的一个非常重要的优势：灵活。对于 viewport 解决 1px 问题、高清方案适配这样的场景， vw/vh 是束手无策的。这又为我们的布局单位选型提供了一个新的依据。

}