04 屏幕适配方案选择

更新时间：2019-07-31 14:36:56

> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。
>
> —— 法布尔

Chrome DevTools 模拟移动设备

使用 Chrome DevTools 的 Device Mode 功能，可以大致了解你的页面在移动设备上呈现的外观和效果，这个是一种非常方便的调试移动端页面的方法。但是需要注意的是，Device Mode 的效果和真机的效果也不是100%一样，所以当你的页面在 Device Mode 测试完成时，别忘了在真机上在看看，确保万无一失。点击 Toggle Device Toolbar 切换设备工具栏，可以打开用于模拟移动设备视口的界面。

移动设备视口模式

要模拟特定移动设备的尺寸，请从 Device 列表中选择一个合适模拟设备。想要立即生效，可以刷新一下浏览器。

限制网络流量

我们知道，移动端的网络状况是错综复杂的，当然我们在开发移动页面时也要兼容这种情况，Chrome DevTools给我们提供了模拟各种网速的功能：

或者按 Command+Shift+P (Mac) 或 Control+Shift+P（Windows、Linux、Chrome 操作系统），以打开命令菜单，输入 3G，然后选择 Enable fast 3G throttling 或 Enable slow 3G throttling。其他的面板功能比如 Sources，Network 等等和PC端的使用方法一致，在这里就不过多解释了。另外在 Application 里有 Service Worker 的相关配置和功能将会在后面的章节讲解。

rem和vw适配方案比较

我们此次的实战项目，主要是一个移动 Web 项目，那么就不得不提屏幕适配。所谓屏幕适配，说白了就是让我们的移动页面在不同的移动端屏幕下，都能展示出良好的效果，那么当下比较流行的适配方案主要有2个：

- rem 适配方案
- vw 适配方案

这两种适配方案既可以搭配使用，也可以结合 Media Query(媒体查询) 来使用，达到针对不同屏幕的适配效果。下面来讲讲两者的区别：

rem适配方案

rem（font size of the root element）是指相对于根元素的字体大小的单位。简单的说它就是一个相对单位。看到rem大家一定会想起em单位，em（font size of the element）是指相对于父元素的字体大小的单位。它们之间其实很相似，只不过计算的规则一个是依赖根元素，一个是依赖父元素。

> rem的适配原理：将我们之前写px的单位换成rem单位，然后动态设置根元素html的font-size大小，从而达到适配的目的。

浏览器 html 默认字体大小（在不修改浏览器字体情况下）是 16px，所以默认情况下 1rem = 16px，但是我们通常需要动态设置 html 的 font-size 。

```
/*使用 MediaQuery 动态设置 html 的 font-size：*/
@media screen and (max-width:360px) and (min-width:321px){
  html{
    font-size:22px;
  }
}

@media screen and (max-width:320px){
  html{
    font-size:30px;
  }
}
```

```
// 使用 Javascript 动态设置 html 的 font-size：
window.addEventListener('resize',function(){
  // 获取视窗宽度
  let htmlWidth = document.documentElement.clientWidth || document.body.clientWidth;
  //获取html
  let htmlDom = document.getElementsByTagName('html')[0];
  htmlDom.style.fontSize = htmlWidth / 10 + 'px';  //求出基准值
})
```

实际应用当中，大多数采用的 Javascript 来动态设置。设置完之后，我们就可以直接利用 rem 单位来给我们的 div 或者其他元素设置宽高等一些属性了，这里就有一个问题，我们一般拿到的UI稿提供的px，怎么转换成我们的rem单位呢？

UI 给的视觉稿一般是以 Iphone6 的屏幕设计，这里一个按钮的标注的宽高是 20*40 单位是 px，由此我们可以得到：

- Iphone6 的屏幕是 375*667 单位是 px（这个单位可以在使用 Chrome DevTools 时得到）；
- 根据上面 JavaScript 设置的 html 的 font-size 得到是 37.5px；
- 那么根据 1rem=37.5px，得到 20px=0.53rem，40px=1.06rem。

```
.button {
  width: 0.53rem;
  height: 1.06rem;
  background-color: red;
}
```

当然，在实际项目中，我们可以使用 sass 的公式来解决这些繁琐的换算问题：

```
@function px2rem($px){
  $rem : 37.5px;
  @return ($px/$rem) + rem;
}

.button {
  width: px2rem(20);
  height: px2rem(40);
  background-color: red;
}
```

好了，关于 rem 的相关知识，我们先讲到这里。如果大家想了解更多，可以去看一下这个慕课网课程：移动web开发适配秘籍Rem。

vw适配方案

vw 其实也是一个 CSS 单位，类似的还有 vh,vmin,vmax 共四个单位，这些单位伴随着 CSS3 的出现就已经有了。但是当时移动 Web 的浪潮已经来临，并且 rem 出现的要早，所以很多人忽略了这个。但是 rem 使用适配是要依赖 Javascript 来进行处理（动态设置 html 根元素的 font-size），而vw 适配方案完全基于 CSS 自身。

- vw : 1vw 等于视口宽度的1%
- vh : 1vh 等于视口高度的1%
- vmin : 选取 vw 和 vh 中最小的那个
- vmax : 选取 vw 和 vh 中最大的那个

视口宽度：表示你在 meta 里设置的那个宽度：

```
<meta name="viewport" content="width=device-width,">
```

使用 document.documentElement.clientWidth 可以获取到浏览器的视口大小，这里要注意不一样的是类似 window.innerWidth 或者 window.screen.width 这些拿到的是浏览器的物理宽度，当width!=device-width时是不等效的哦。

同理我们将Iphone6的UI搞下，这里一个按钮的标注的宽高是 20*40 单位是 px，转换成 vw：

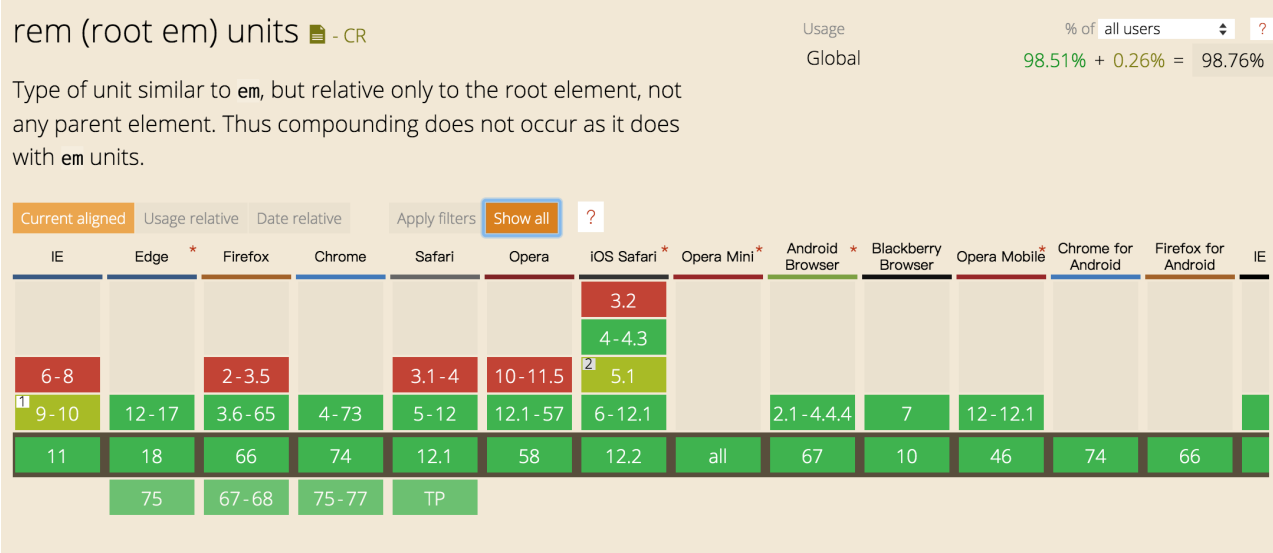
```
.button {
  width: 5.3vw; (20/375*100vw)
  height: 10.6vw; (40/375*100vw)
  background-color: red;
}
```

同理，你可能也需要一个sass公式来解决繁琐的换算问题：

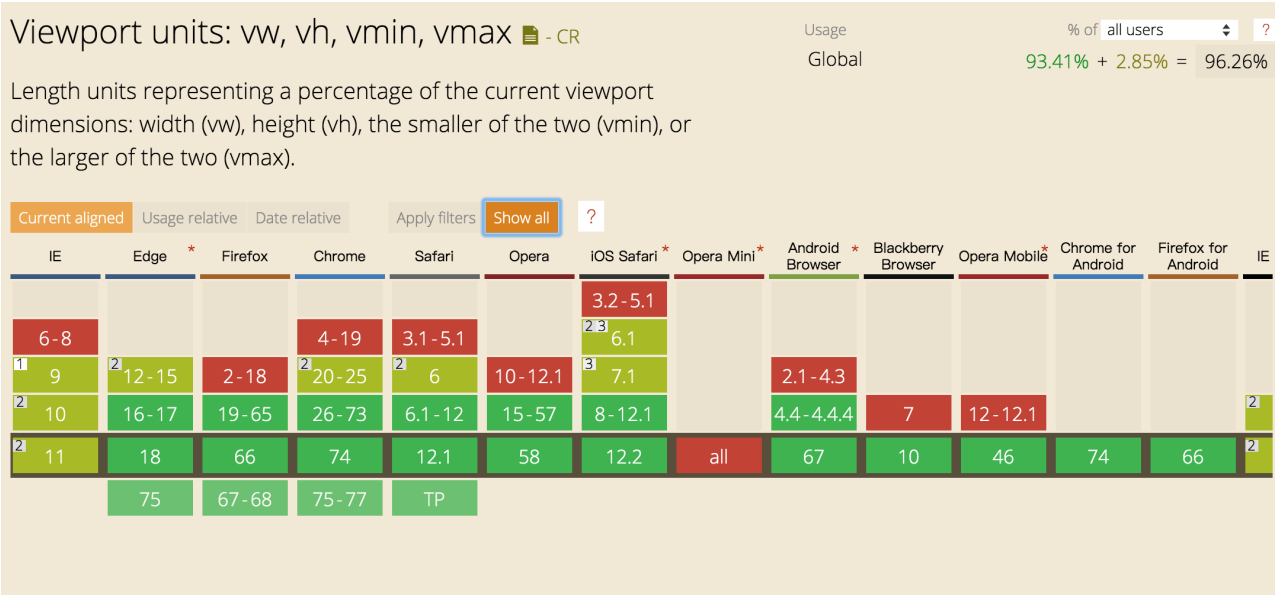
```
@function vw($px) {
  @return ($px / 375) * 100vw;
}
```

那么 vw 方案相比 rem 优势在于，不需要 JavaScript 来动态设置 html 根元素的 font-size；劣势在于浏览器兼容性不如 rem。

rem兼容性：



vw兼容性：



由于现在市面上 Android4.4 以下的机型还占有一定的比例（9%左右），这个量还是不容忽视的，这也是现在 vw 适配方案没有 rem 流行的主要原因之一。好在既然所有最新浏览器都已经支持 vw，那么随着时间推移，相信 vw 未来必将会流行。

postcss-px-to-viewport适配

前面说了这么多理论知识，那么下面就回到我们的实战项目中。本次实战项目采用 vw 适配方案（因为 rem 的适配方案用的太多了，尝试一下新的 vw 方案），使用 postcss 的 postcss-px-to-viewport 插件来帮我们进行px和vw之间的转换。

```
# 安装postcss-px-to-viewport：
npm install postcss-px-to-viewport --save
```

在 postcss.config.js 添加配置：

```
"postcss-px-to-viewport": {
  viewportWidth: 375,  // 视窗的宽度，对应的是我们设计稿的宽度，Iphone6的一般是375 （xx/375*100vw）
  viewportHeight: 667, // 视窗的高度，Iphone6的一般是667
  unitPrecision: 3,    // 指定`px`转换为视窗单位值的小数位数（很多时候无法整除）
  viewportUnit: "vw",  // 指定需要转换成的视窗单位，建议使用vw
  selectorBlackList: ['.ignore', '.hairlines'],// 指定不转换为视窗单位的类，可以自定义，可以无限添加,建议定义一至两个通用的类名
  minPixelValue: 1,    // 小于或等于`1px`不转换为视窗单位，你也可以设置为你想要的值
  mediaQuery: false    // 允许在媒体查询中转换`px`
},
```

设置之后，我们在项目里直接写 px 单位，然后 postCss 就可以帮我们转换成vw单位，很方便有木有。

```
.name-info {
  position: absolute;
  right: 12px;
  top: 273px;
  border-radius: 5px;
  display: flex;
}
```

实际效果：

```
.name-info[data-v-354fcfd8] {
  position: absolute;
  right: 3.2vw;
  top: 72.8vw;
  border-radius: 1.333vw;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
}
```

同时，我们把autoprefixer也默认配置上去：

```
module.exports = {
  plugins: {
    autoprefixer: {},
    'postcss-px-to-viewport': {
      ***
    }
}
```

小结

本章节主要给大家介绍了 Chrome DevTools 模拟移动设备和移动端的常见适配方案，包括 rem 适配和 vw 适配，同时比较了两者的差别和优劣。另外在项目里引入 postcss-px-to-viewport 来帮助我们解决单位转换问题。

}

03 开发环境准备

05 初始化前端项目