如何在Vue项目中使用vw实现移动端适配(转)

BULL_DEBUG 关注
5 2018.03.16 09:59:49 字数 4,368 阅读 17,875

有关于移动端的适配布局一直以来都是众说纷纭，对应的解决方案也是有很多种。在《使用Flexible实现手淘H5页面的终端适配》提出了Flexible的布局方案，随着 `viewport` 单位越来越受到众多浏览器的支持，因此在《再聊移动端页面的适配》一文中提出了 `vw` 来做移动端的适配问题。到目前为止不管是哪一种方案，都还存在一定的缺陷。言外之意，还没有哪一个方案是完美的。

事实上真的不完美？其实不然。最近为了新项目中能更完美的使用 `vw` 来做移动端的适配。探讨出一种能解决不兼容 `viewport` 单位的方案。今天整理一下，与大家一起分享。如果方案中存在一定的缺陷，欢迎大家一起拍正。

准备工作

对于Flexible或者说 `vw` 的布局，其原理不在这篇文章进行阐述。如果你想追踪其中的原委，强烈建议你阅读早前整理的文章《使用Flexible实现手淘H5页面的终端适配》和《再聊移动端页面的适配》。

> 说句题外话，由于Flexible的出现，也造成很多同学对 `rem` 的误解。正如当年大家对 `div` 的误解一样。也因此，大家都觉得 `rem` 是万能的，他能直接解决移动端的适配问题。事实并不是如此，至于为什么，我想大家应该去阅读 `flexible.js` 源码，我相信你会明白其中的原委。

回到我们今天要聊的主题，怎么实现 `vw` 的兼容问题。为了解决这个兼容问题，我将借助Vue官网提供的构建工程以及一些PostCSS插件来完成。在继续后面的内容之前，需要准备一些东西：

- NodeJs
- NPM
- Webpack
- Vue-cli
- postcss-import
- postcss-url
- postcss-aspect-ratio-mini
- postcss-cssnext
- autoprefixer
- postcss-px-to-viewport
- postcss-write-svg
- cssnano
- postcss-viewport-units
- Viewport Units Buggyfill

对于这些起什么作用，先不阐述，后续我们会聊到上述的一些东西。

使用Vue-cli来构建项目

对于NodeJs、NPM和Webpack相关介绍，大家可以查阅其对应的官网，这里默认你的系统环境已

BULL_DEBUG 关注
总资产9 (约0.83元)

推荐阅读
前端面试每日 3+1 —— 第239天
阅读 2,351
高级前端面试题目大全(一)
阅读 26,305
map中使用await 异步函数
阅读 82
2019年回顾和2020年展望
阅读 830
理解原型和原型链
阅读 327

使用Vue-cli构建项目

为了不花太多的时间去深入的了解Webpack（Webpack对我而言，太蛋疼了），所以我直接使用Vue-cli来构建自己的项目，因为我一般使用Vue来做项目。如果你想深入的了解Webpack，建议你阅读下面的文章：

- Webpack文档
- Awesome Webpack
- Webpack 教程资源收集
- Vue+Webpack开发可复用的单页面富应用教程

接下来的内容，直接使用Vue官方提供的Vue-cli的构建工具来构建Vue项目。首先需要安装Vue-cli：

```
$ npm install -g vue-cli
```

全局先安装Vue-cli，假设你安装好了Vue-cli。这样就可以使用它来构建项目：

```
vue init webpack vw-layout
```

根据命令提示做相应的操作：

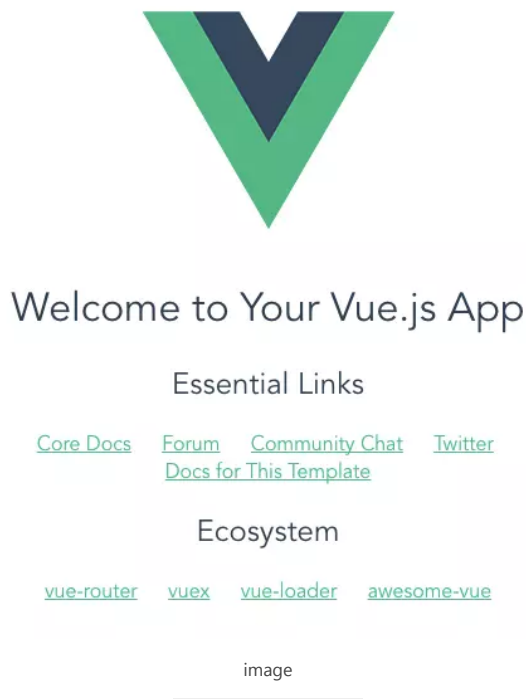
```
➜ weihua@liaoairendeMacBook-Air  /Applications/XAMPP/htdocs/vue-app  vue init webpack vw-layout

? Project name vw-layout
? Project description vw-layout
? Author airenliao@gmail.com
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm

   vue-cli · Generated "vw-layout".


# Installing project dependencies ...
# ========================


> fsevents@1.1.3 install /Applications/XAMPP/xamppfiles/htdocs/vue-app/vw-layout/node_modules/fsevents
> node install

[fsevents] Success: "/Applications/XAMPP/xamppfiles/htdocs/vue-app/vw-layout/node_modules/fsevents/lib/bin
ding/Release/node-v59-darwin-x64/fse.node" is installed via remote

> uglifyjs-webpack-plugin@0.4.6 postinstall /Applications/XAMPP/xamppfiles/htdocs/vue-app/vw-layout/node_m
odules/webpack/node_modules/uglifyjs-webpack-plugin
> node lib/post_install.js

npm notice created a lockfile as package-lock.json. You should commit this file.
added 1258 packages in 77.771s

# Project initialization finished!
# ========================

To get started:

  cd vw-layout
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack
```

image

进入到刚创建的 `vw-layout`：

```
cd vw-layout
```

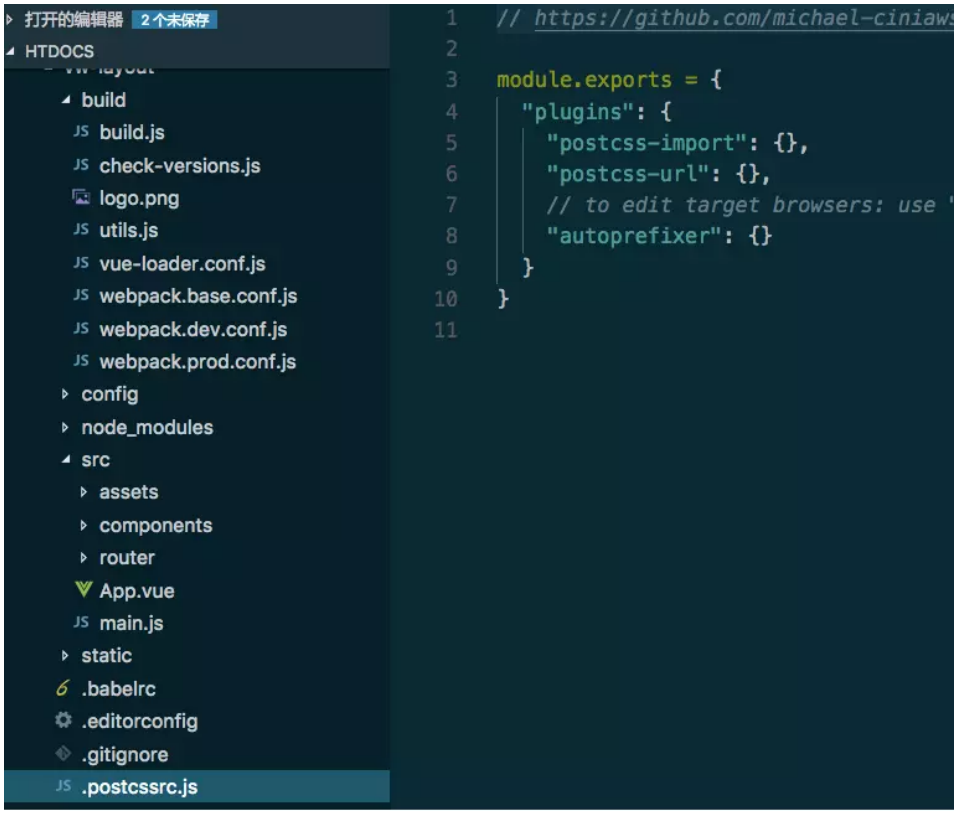
在浏览器执行 `http://localhost:8080` ，就可以看以默认的页面效果：



image

> 以前的版本需要先执行 `npm i` 安装项目需要的依赖关系。现在新版本的可以免了。

这时，可以看到的项目结构如下：



使用Vue-cli构建项目

通过Vue-cli构建的项目，在项目的根目录下有一个 `.postcssrc.js` ，默认情况下已经有了：

```
module.exports = {
    "plugins": {
        "postcss-import": {},
        "postcss-url": {},
        "autoprefixer": {}
    }
}

```

对应我们开头列的的PostCSS插件清单，现在已经具备了：

- postcss-import
- postcss-url
- autoprefixer

简单的说一下这几个插件。

postcss-import

`postcss-import` 相关配置可以点击这里。目前使用的是默认配置。只在 `.postcssrc.js` 文件中引入了该插件。

`postcss-import` 主要功有是解决 `@import` 引入路径问题。使用这个插件，可以让你很轻易的使用本地文件、 `node_modules` 或者 `web_modules` 的文件。这个插件配合 `postcss-url` 让你引入文件变得更轻松。

postcss-url

`postcss-url` 相关配置可以点击这里。该插件主要用来处理文件，比如图片文件、字体文件等引用路径的处理。

在Vue项目中， `vue-loader` 已具有类似的功能，只需要配置中将 `vue-loader` 配置进去。

autoprefixer

`autoprefixer` 插件是用来自动处理浏览器前缀的一个插件。如果你配置了 `postcss-cssnext` ，其中就已具备了 `autoprefixer` 的功能。在配置的时候，未显示的配置相关参数的话，表示使用的是Browserslist指定的列表参数，你也可以像这样来指定 `last 2 versions` 或者 `> 5%` 。

如此一来，你在编码时不再需要考虑任何浏览器前缀的问题，可以专心撸码。这也是PostCSS最常用的一个插件之一。

其他插件

Vue-cli默认配置了上述三个PostCSS插件，但我们要完成 `vw` 的布局兼容方案，或者说让我们能更专心的撸码，还需要配置下面的几个PostCSS插件：

- postcss-aspect-ratio-mini
- postcss-px-to-viewport
- postcss-write-svg
- postcss-cssnext
- cssnano

```
npm i postcss-aspect-ratio-mini postcss-px-to-viewport postcss-write-svg postcss-cssnext postcss-
```

安装成功之后，在项目根目录下的 `package.json` 文件中，可以看到新安装的依赖包：

```
"dependencies": {
    "cssnano": "^3.10.0",
    "postcss-aspect-ratio-mini": "0.0.2",
    "postcss-cssnext": "^3.1.0",
    "postcss-px-to-viewport": "0.0.3",
    "postcss-viewport-units": "^0.1.3",
    "postcss-write-svg": "^3.0.1",
    "vue": "^2.5.2",
    "vue-router": "^3.0.1"
},
```

接下来在 `.postcssrc.js` 文件对新安装的PostCSS插件进行配置：

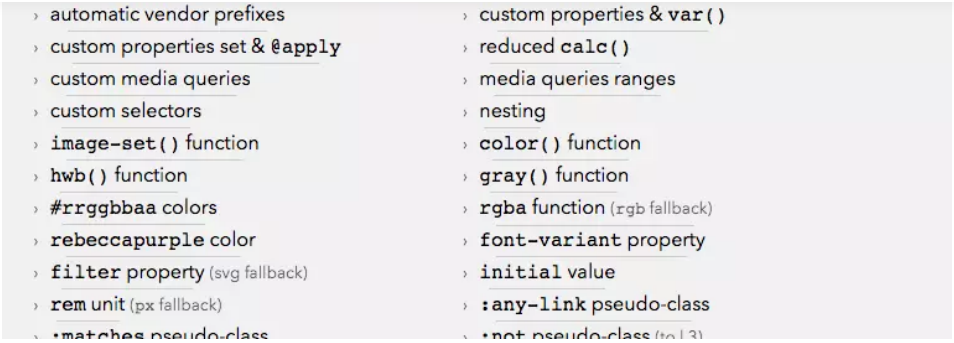
```
module.exports = {
    "plugins": {
        "postcss-import": {},
        "postcss-url": {},
        "postcss-aspect-ratio-mini": {},
        "postcss-write-svg": {
            utf8: false
        },
        "postcss-cssnext": {},
        "postcss-px-to-viewport": {
            viewportWidth: 750,     // (Number) The width of the viewport.
            viewportHeight: 1334,    // (Number) The height of the viewport.
            unitPrecision: 3,       // (Number) The decimal numbers to allow the REM units to grc
            viewportUnit: 'vw',     // (String) Expected units.
            selectorBlackList: ['.ignore', '.hairlines'],  // (Array) The selectors to ignore and
            minPixelValue: 1,       // (Number) Set the minimum pixel value to replace.
            mediaQuery: false       // (Boolean) Allow px to be converted in media queries.
        },
        "postcss-viewport-units":{},
        "cssnano": {
            preset: "advanced",
            autoprefixer: false,
            "postcss-zindex": false
        }
    }
}
```

> **特别声明：**由于 `cssnext` 和 `cssnano` 都具有 `autoprefixer` ,事实上只需要一个，所以把默认的 `autoprefixer` 删除掉，然后把 `cssnano` 中的 `autoprefixer` 设置为 `false` 。对于其他的插件使用，稍后会简单的介绍。

由于配置文件修改了，所以重新跑一下 `npm run dev` 。项目就可以正常看到了。接下来简单的介绍一下后面安装的几个插件的作用。

postcss-cssnext

`postcss-cssnext` 其实就是cssnext。该插件可以让我们使用CSS未来的特性，其会对这些特性做相关的兼容性处理。其包含的特性主要有：



postcss-cssnext

有关于 cssnext 的每个特性的操作文档，可以点击这里浏览。

cssnano

cssnano 主要用来压缩和清理CSS代码。在Webpack中， cssnano 和 css-loader 捆绑在一起，所以不需要自己加载它。不过你也可以使用 postcss-loader 显式的使用 cssnano 。有关于 cssnano 的详细文档，可以点击这里获取。

在 cssnano 的配置中，使用了 preset: "advanced" ，所以我们需要另外安装：

```
npm i cssnano-preset-advanced --save-dev
```

cssnano 集成了一些其他的PostCSS插件，如果你想禁用 cssnano 中的某个插件的时候，可以像下面这样操作：

```
"cssnano": {
    autoprefixer: false,
    "postcss-zindex": false
}
```

上面的代码把 autoprefixer 和 postcss-zindex 禁掉了。前者是有重复调用，后者是一个讨厌的东东。只要启用了这个插件， z-index 的值就会重置为 1 。这是一个天坑，**千万记得将 postcss-zindex 设置为 false** 。

postcss-px-to-viewport

postcss-px-to-viewport 插件主要用来把 px 单位转换为 vw 、 vh 、 vmin 或者 vmax 这样的视窗单位，也是 vw 适配方案的核心插件之一。

在配置中需要配置相关的几个关键参数：

```
"postcss-px-to-viewport": {
    viewportWidth: 750,      // 视窗的宽度，对应的是我们设计稿的宽度，一般是750
    viewportHeight: 1334,    // 视窗的高度，根据750设备的宽度来指定，一般指定1334，也可以不配置
    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
    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`
}
```

接在代码中写 px，比如：

```
.test {
    border: .5px solid black;
    border-bottom-width: 4px;
    font-size: 14px;
    line-height: 20px;
    position: relative;
}
[w-188-246] {
    width: 188px;
}

```

编译出来的CSS：

```
.test {
    border: .5px solid #000;
    border-bottom-width: .533vw;
    font-size: 1.867vw;
    line-height: 2.667vw;
    position: relative;
}
[w-188-246] {
    width: 25.067vw;
}

```

在不想要把 px 转换为 vw 的时候，首先在对应的元素（ html ）中添加配置中指定的类名 .ignore 或 .hairlines (.hairlines 一般用于设置 border-width:0.5px 的元素中)：

```
<div class="box ignore"></div>

```

写CSS的时候：

```
.ignore {
    margin: 10px;
    background-color: red;
}
.box {
    width: 180px;
    height: 300px;
}
.hairlines {
    border-bottom: 0.5px solid red;
}

```

编译出来的CSS:

```
.box {
    width: 24vw;
    height: 40vw;
}
.ignore {
    margin: 10px; /*.box元素中带有.ignore类名，在这个类名写的`px`不会被转换*/
    background-color: red;
}
.hairlines {
    border-bottom: 0.5px solid red;
}

```

- 容器适配，可以使用 vw
- 文本的适配，可以使用 vw
- 大于 1px 的边框、圆角、阴影都可以使用 vw
- 内距和外距，可以使用 vw

postcss-aspect-ratio-mini

postcss-aspect-ratio-mini 主要用来处理元素容器宽高比。在实际使用的时候，具有一个默认的结构

```
<div aspectratio>
    <div aspectratio-content></div>
</div>

```

在实际使用的时候，你可以把自定义属性 aspectratio 和 aspectratio-content 换成相应的类名，比如：

```
<div class="aspectratio">
    <div class="aspectratio-content"></div>
</div>

```

我个人比较喜欢用自定义属性，它和类名所起的作用是同等的。结构定义之后，需要在你的样式文件中添加一个统一的宽度比默认属性：

```
[aspectratio] {
    position: relative;
}
[aspectratio]::before {
    content: '';
    display: block;
    width: 1px;
    margin-left: -1px;
    height: 0;
}

[aspectratio-content] {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    width: 100%;
    height: 100%;
}

```

如果我们想要做一个 188:246 （ 188 是容器宽度， 246 是容器高度）这样的比例容器，只需要这样使用：

```
[w-188-246] {
    aspect-ratio: '188:246';
}

```

有一点需要特别注意： aspect-ratio 属性不能和其他属性写在一起，否则编译出来的属性只会留

编译前的CSS如下：

```
[w-188-246] {
    width: 188px;
    background-color: red;
    aspect-ratio: '188:246';
}

```

编译之后：

```
[w-188-246]:before {
    padding-top: 130.85106382978725%;
}

```

主要是因为在插件中做了相应的处理，不在每次调用 `aspect-ratio` 时，生成前面指定的默认样式代码，这样代码没那么冗余。所以在使用的时候，需要把 `width` 和 `background-color` 分开来写：

```
[w-188-246] {
    width: 188px;
    background-color: red;
}
[w-188-246] {
    aspect-ratio: '188:246';
}

```

这个时候，编译出来的CSS就正常了：

```
[w-188-246] {
    width: 25.067vw;
    background-color: red;
}
[w-188-246]:before {
    padding-top: 130.85106382978725%;
}

```

有关于宽高比相关的详细介绍，如果大家感兴趣的话，可以阅读下面相关的文章：

- CSS实现长宽比的几种方案
- 容器长宽比
- Web中如何实现纵横比
- 实现精准的流体排版原理

> 目前采用PostCSS插件只是一个过渡阶段，在将来我们可以直接在CSS中使用 `aspect-ratio` 属性来实现长宽比。

postcss-write-svg

`postcss-write-svg` 插件主要用来处理移动端 `1px` 的解决方案。该插件主要使用的是 `border-image` 和 `background` 来做 `1px` 的相关处理。比如：

```
    @rect {
        fill: var(--color, black);
        width: 100%;
        height: 50%;
    }
}
.example {
    border: 1px solid transparent;
    border-image: svg(1px-border param(--color #00b1ff)) 2 2 stretch;
}
```

编译出来的CSS:

```
.example {
    border: 1px solid transparent;
    border-image: url("data:image/svg+xml;charset=utf-8,%3Csvg xmlns='http://www.w3.org/2000/svg'
}
```

上面演示的是使用 `border-image` 方式，除此之外还可以使用 `background-image` 来实现。比如：

```
@svg square {
    @rect {
        fill: var(--color, black);
        width: 100%;
        height: 100%;
    }
}

#example {
    background: white svg(square param(--color #00b1ff));
}
```

编译出来就是：

```
#example {
    background: white url("data:image/svg+xml;charset=utf-8,%3Csvg xmlns='http://www.w3.org/2000/
}
```

解决 `1px` 的方案除了这个插件之外，还有其他的方法。可以阅读前期整理的《再谈Retina下 `1px` 的解决方案》一文。

> **特别声明：**由于有一些低端机对 `border-image` 支持度不够友好，个人建议你使用 `background-image` 的这个方案。

CSS Modules

Vue中的 `vue-loader` 已经集成了CSS Modules的功能，个人建议在项目中开始使用CSS Modules。特别是在Vue和React的项目中，CSS Modules具有很强的优势和灵活性。建议看看CSS In JS相关的资料。在Vue中，使用CSS Modules的相关文档可以阅读Vue官方提供的文档《CSS Modules》。

postcss-viewport-units

到此为止，有关于所需要的PostCSS已配置完。并且简单的介绍了各个插件的作用，至于详细的文档和使用，可以参阅对应插件的官方文档。

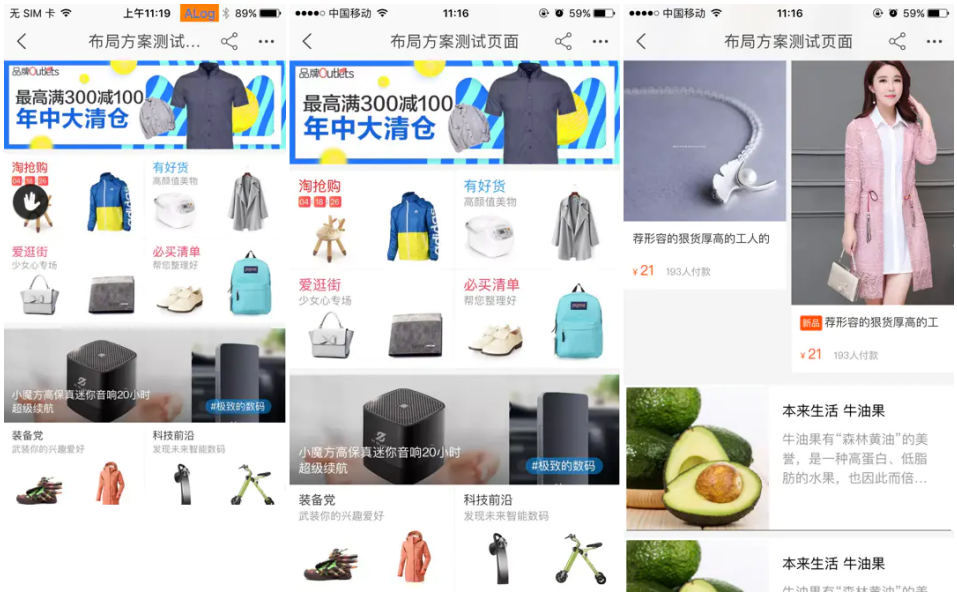
vw兼容方案

在《再聊移动端页面的适配》一文中，详细介绍了，怎么使用 vw 来实现移动端的适配布局。这里不做详细的介绍。建议你花点时间阅读这篇文章。

先把未做兼容处理的示例二维码贴一个：

image

你可以使用手淘App、优酷APP、各终端自带的浏览器、UC浏览器、QQ浏览器、Safari浏览器和Chrome浏览器扫描上面的二维码，您看到相应的效果：



image

但还有不支持的，比如下表中的 No ，表示的就是不支持

品牌	型号	系统版本	分辨率	屏幕尺寸	手淘APP	优酷APP	原生浏览器	QQ浏览器	UC浏览器	Chrome浏览器
华为	Mate9	Android7.0	1080 x 1920	5英寸	Yes	Yes	No	Yes	Yes	Yes
华为	Mate7	Android4.2	1080 x 1920	5.2英寸	Yes	Yes	No	Yes	Yes	Yes
魅族	Mx4 (M460移动4G)	Android4.4.2	1152 x 1920	5.36英寸	Yes	No	No	Yes	Yes	Yes
Opp	R7007	Android4.3	1280 x 720	5英寸	Yes	No	No	Yes	Yes	No

三星	N9008 (Galaxy Note3)	Android4.4.2	1080 x 1920	5.7英寸	Yes	No	Yes	Yes	Yes	Yes
华硕	ZenFone5(x86)	Android4.3	720 x 280	5英寸	No	No	No	Yes	No	No

正因如此，很多同学都不敢尝这个螃蟹。害怕去处理兼容性的处理。不过不要紧，今天我把最终的解决方案告诉你。

最终的解决方案，就是使用 `viewport` 的polyfill：Viewport Units Buggyfill。使用 `viewport-units-buggyfill` 主要分以下几步走：

引入JavaScript文件

`viewport-units-buggyfill` 主要有两个JavaScript文件： `viewport-units-buggyfill.js` 和 `viewport-units-buggyfill.hacks.js` 。你只需要在你的HTML文件中引入这两个文件。比如在Vue项目中的 `index.html` 引入它们：

```
<script src="//g.alicdn.com/fdilab/lib3rd/viewport-units-buggyfill/0.6.2/??viewport-units-buggyfi
```

你也可以使用其他的在线CDN地址，也可将这两个文件合并压缩成一个 `.js` 文件。这主要看你自己的兴趣了。

第二步，在HTML文件中调用 `viewport-units-buggyfill` ，比如：

```
<script>
    window.onload = function () {
        window.viewportUnitsBuggyfill.init({
            hacks: window.viewportUnitsBuggyfillHacks
        });
    }
</script>
```

为了你Demo的时候能获取对应机型相关的参数，我在示例中添加了一段额外的代码，估计会让你有点烦：

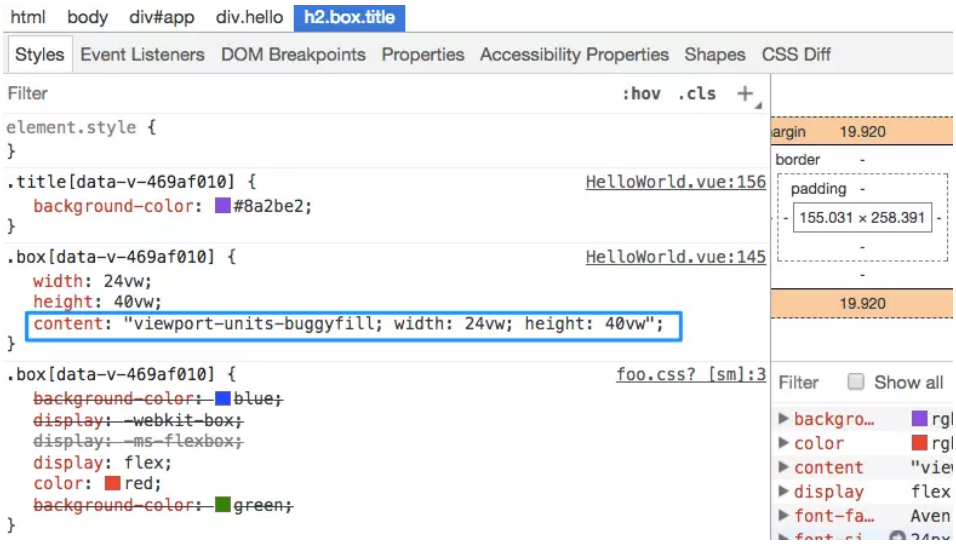
```
<script>
    window.onload = function () {
        window.viewportUnitsBuggyfill.init({
        hacks: window.viewportUnitsBuggyfillHacks
        });

        var winDPI = window.devicePixelRatio;
        var uAgent = window.navigator.userAgent;
        var screenHeight = window.screen.height;
        var screenWidth = window.screen.width;
        var winWidth = window.innerWidth;
        var winHeight = window.innerHeight;

        alert(
            "Windows DPI:" + winDPI +
            ";\ruAgent:" + uAgent +
            ";\rScreen Width:" + screenWidth +
            ";\rScreen Height:" + screenHeight +
            ";\rWindow Width:" + winWidth +
            ";\rWindow Height:" + winHeight
        )
    }
```

要在样式中添加 `content`：

```
.my-viewport-units-using-thingie {
    width: 50vmin;
    height: 50vmax;
    top: calc(50vh - 100px);
    left: calc(50vw - 100px);

    /* hack to engage viewport-units-buggyfill */
    content: 'viewport-units-buggyfill; width: 50vmin; height: 50vmax; top: calc(50vh - 100px); l
}
```

这可能会令你感到恶心，而且我们不可能每次写 `vw` 都去人肉的计算。特别是在我们的这个场景中，咱们使用了 `postcss-px-to-viewport` 这个插件来转换 `vw`，更无法让我们人肉的去添加 `content` 内容。

这个时候就需要前面提到的 `postcss-viewport-units` 插件。这个插件将让你无需关注 `content` 的内容，插件会自动帮你处理。比如插件处理后的代码：



image

Viewport Units Buggyfill还提供了其他的功能。详细的这里不阐述了。但是 `content` 也会引起一定的副作用。比如 `img` 和伪元素 `::before`（`:before`）或 `::after` （`:after`）。在 `img` 中 `content` 会引起部分浏览器下，图片不会显示。这个时候需要全局添加：

```
img {
    content: normal !important;
}
```

而对于 `::after` 之类的，就算是里面使用了 `vw` 单位，Viewport Units Buggyfill对其并不会起作用。比如：

```
// 编译前
.after {
    content: 'after content';
    display: block;
    width: 100px;
    height: 20px;
    background: green;
}
```

```
    width: 13.333vw;
    height: 2.667vw;
    background: green;
}
```

这个时候我们需要通过添加额外的标签来替代伪元素（这个情景我没有测试到，后面自己亲测一下）。

到了这个时候，你就不需要再担心兼容问题了。比如下面这个示例：

image

请用你的手机，不管什么APP扫一扫，你就可以看到效果。（小心弹框哟），如果你发现了还是有问题，请把弹出来的信息截图发给我。

如查你想看看别的机型效果，可以点击这里、这里、这里、还有这里。整个示例的源码，可以点击这里下载。

> 如果你下载了示你源码，先要确认你的系统环境能跑Vue的项目，然后下载下来之后，解压缩，接着运行 `npm i` ，再运行 `npm run dev` ，你就可以看到效果了。

总结

如果你看到这里了，希望这篇文章对你有所帮助。能帮助你解决项目中的实际问题，让你不再担心移动端的适配问题。当然更希望的是你在实际的项目中用起这个方案，把碰到的问题及时反馈给偶。如果你有更好的方案，欢迎在下面的评论中与我们一起分享。

写下你的评论...

全部评论 7 只看作者 按时间倒序 按时间正序

Chris
6楼 02.13 12:04
其实就一个插件...
赞 回复

晶哥哥的号
5楼 2019.10.09 12:27
这也太多了吧，装那么多插件 晕倒了
赞 回复

时光匆匆_e728
4楼 2019.07.23 12:33
使用插件后为什么有的组件中的z-index，设置的是3888，出来显示的2?
赞 回复

时光匆匆_e728
2019.07.23 17:22
解决Z-index失效问题了
"cssnano": {
"cssnano-preset-advanced": {
zindex: false,
autoprefixer: false
},
}
cssnano升级了，其中配置的参数有变化。
回复

添加新评论

fish_3583
3楼 2019.07.21 19:34
受教了，谢谢分享
赞 回复

咖啡教室
2楼 2019.07.08 16:12
改了.postcsssrc.js文件一运行就报错，愁死我了：Refused to load the image 'http://localhost:8080/favicon.ico' because it violates the following Content Security Policy directive: "default-src 'none'". Note that 'img-src' was not explicitly set, so 'default-src' is used as a fallback.
赞 回复

回复

添加新评论

被以下专题收入，发现更多相似内容

Vue Vue 前端 javascript 短板不足真TM多 vue学习 Vue 展开更多

推荐阅读

更多精彩内容 >

[分享] [New]Vue项目使用vw实现移动端适配教程

正文开始 使用vue-cli新建项目 安装依赖 复制以下代码： 复制进行 postcssrc.js 配置.post...

大象无痕 阅读 14,341 评论 23 赞 14

从搭建vue-脚手架到掌握webpack配置（三.多页面构建）

前言 上一期中我们通过引入了插件实现了不少功能——样式抽离、公共模块提取、代码压缩等等；本期开始讲讲可能会用到的第...

JasonWild 阅读 1,199 评论 0 赞 8

Vue相关开源项目库汇总(史上最全)

目录 UI组件 开发框架 实用库 服务端 辅助工具 应用实例 Demo示例 UI组件 element★13489 ...

余生社会 阅读 15,343 评论 7 赞 229

去看歌舞团

哈哈，说是看歌舞团，实际上是带我去见他的朋友啥了的，嘿嘿，总之没给他丢人，不错不错，以后我要一直保持美丽，耶，今...

丸那个丸子 阅读 27 评论 0 赞 0

离别祝福（补记）

正月十五，一共事近五年的同事在微信群中与大家告别。一时间，众人纷纷表达感谢，祝愿之情，满屏的温暖。有感而发，作小诗...

ideal_dan 阅读 33 评论 0 赞 0

写下你的评论... 评论7 赞68 ···