开门见山，本篇将总结一下 MobileWeb 的适配方法，即我们常说的H5页面、手机页面、WAP页、webview页面等等。

本篇讨论的页面指***专门针对手机设备设计***的页面，并非兼容全设备的响应式布局。 文中提到的 ***device-width*** 指 viewport meta 标签中 width 的值，即由浏览器指定的值，常用机型对应值可参照[Screen Sizes](http://screensiz.es/phone)。

**适配达到的效果是什么？**

在不同尺寸的手机设备上，页面“**相对性的达到合理的展示（自适应）**”或者“**保持统一效果的等比缩放（看起来差不多）**”。

**适配应关注哪些要素？**

一般来说，我们需要关注的是：字体、高宽间距、图像（图标、图片）。

其中，图像相对要复杂一些，针对流量、清晰度等问题网上也有比较成熟的解决方案，比如：矢量化、字体化、image-set 等等，在此不做深入。在满足快速开发的需求下，我们使用较为偷懒的方式：**利用 css 将图像限定在元素内（ img 图片使用**[max-]width: 100%**，背景图像使用 *background-size* ），布局只针对元素进行**。

另外要考虑到，设计师设计视觉稿时使用什么样的宽度，才能既满足设计自身的需求又能让前端开发方便的切图适配。

**举个例子**

围绕这三要素，我们用一个小例子来说明接下来要介绍的三种方案的实现方式，按 640px 标准需实现的效果如图：



**固定高度，宽度自适应**

这是目前最通用的一种做法，属于自适应布局，viewport width 设置为 *device-width*，以较小宽度（如 320px）的视觉稿作为参照进行布局。垂直方向的高度和间距使用定值，水平方向混合使用定值和百分比或者利用弹性布局，最终达到“当手机屏幕变化时，横向拉伸或者填充空白的效果”。图像元素根据容器情况，使用定值或者 *background-size* 缩放。

粗略浏览了下一些大厂的首页，像百度、腾讯、Facebook、Twitter 都是采用的这种方案。

**要点：**

* 以小宽度作为参照是因为如果布局满足了小宽度的摆放，当屏幕变宽时，简单的填充空白就可以了；而如果反过来就可能造成“挤坏了”，考虑 header 区域，左测 logo 右测横向 nav 的情况。
* 需要小宽度的布局，又需要大宽度的图像，这是一个矛盾点。
* 320px 过于窄小，不利于页面的设计；只能设计横向拉伸的元素布局，存在很多局限性。
* 兼容性较好。

实现比较简单，样式中的尺寸都按照视觉稿二分之一大小设置，[查看效果和代码](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-fixed-height.html)。

**固定宽度，viewport 缩放**

视觉稿、页面宽度、viewport width 使用统一宽度，利用浏览器自身缩放完成适配。页面样式（包括图像元素）完全按照视觉稿的尺寸，使用定值单位 （px、em）即可完成。

**优点：**

* ***开发简单***    缩放交给浏览器，完全按视觉稿切图。
* ***还原精准***    绝对等比例缩放，可以精准还原视觉稿（不考虑清晰度的情况下）。
* ***测试方便***    在PC端即可完成大部分测试，手机端只需酌情调整一些细节（比如图标、字体混合排列时，因为字体不同造成的对齐问题）。

**存在的问题：**

* ***像素丢失***     对于一些分辨率较低的手机，可能设备像素还未达到指定的 viewport 宽度，此时屏幕的渲染可能就不准确了。比较常见的是边框“消失”了，不过随着手机硬件的更新，这个问题会越来越少的。
* ***缩放失效***     某些安卓机不能正常的根据 meta 标签中 *width* 的值来缩放 viewport，需要配合 *initial-scale* 。
* ***文本折行***    存在于缩放失效的机型中，某些手机为了便于文本的阅读，在文本到达 viewport 边缘（非元素容器的边缘）时即进行折行，而当 viewport 宽度被修正后，浏览器并没有正确的重绘，所以就发现文本没有占满整行。一些常用的段落性文本标签会存在该问题。

缩放失效问题需通过 js 动态设定 *initial-scale*。

var fixScreen = function() {

var metaEl = doc.querySelector('meta[name="viewport"]'),

metaCtt = metaEl ? metaEl.content : '',

matchScale = metaCtt.match(/initial\-scale=([\d\.]+)/),

matchWidth = metaCtt.match(/width=([^,\s]+)/);

if ( metaEl && !matchScale && ( matchWidth && matchWidth[1] != 'device-width') ) {

var width = parseInt(matchWidth[1]),

iw = win.innerWidth || width,

ow = win.outerWidth || iw,

sw = win.screen.width || iw,

saw = win.screen.availWidth || iw,

ih = win.innerHeight || width,

oh = win.outerHeight || ih,

ish = win.screen.height || ih,

sah = win.screen.availHeight || ih,

w = Math.min(iw,ow,sw,saw,ih,oh,ish,sah),

scale = w / width;

if ( ratio < 1) {

metaEl.content += ',initial-scale=' + ratio + ',maximum-scale=' + ratio + ', minimum-scale=' + scale;

}

}

}

文本折行问题可以通过 css 样式解决。

section, p, div,

h1, h2, h3, h4, h5, h6,

.fix-break {

background: tranparent url('about:blank');

word-break: break-all;

}

既然该方案使用固定宽度值，那么这个值是多少合适呢？首要考虑的是主流分辨率，可参考 [Screen Sizes](http://screensiz.es/phone) 和 [友盟指数](http://www.umindex.com/devices/android_resolutions) 的数据；其次要考虑设计部门常用的设计尺寸，综合协调，最终确定一个合适的值。

该处使用 640px 来实现例子，[查看效果和代码](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-fixed-width.html)。

**利用 rem 布局**

依照某特定宽度设定 rem 值（即 html 的 *font-size*），页面任何需要弹性适配的元素，尺寸均换算为 rem 进行布局；当页面渲染时，根据页面有效宽度进行计算，调整 rem 的大小，动态缩放以达到适配的效果。利用该方案，还可以根据 *devicePixelRatio* 设定 *initial-scale* 来放大 viewport，使页面按照物理像素渲染，提升清晰度。

**优点：**

* 清晰度高，能达到物理像素的清晰度。
* 能解决 DPR 引起的“1像素”问题。
* 向后兼容较好，即便屏幕宽度增加、PPI 增加该方案依旧适用。

**缺点：**

* 适配 js 需尽可能早进入，减少（避免）viewport 变化引起的重绘。
* 某些Android机会丢掉 rem 小数部分。
* 需要预编译库进行单位转换。

开发时，css 及 js 都以 16px 作为基数换算 rem，借助预编译库（以 scss 为例），我们设定一个动态尺寸单位 $ppr: 750px/16px/1rem ，即 pixel per rem，任何使用弹性尺寸的地方写作：width: 100px/$ppr。

动态调整 rem 的方法如下：

var fixScreen = function() {

var metaEl = doc.querySelector('meta[name="viewport"]'),

metaCtt = metaEl ? metaEl.content : '',

matchScale = metaCtt.match(/initial\-scale=([\d\.]+)/),

matchWidth = metaCtt.match(/width=([^,\s]+)/);

if ( !metaEl ) { // REM

var docEl = doc.documentElement,

maxwidth = docEl.dataset.mw || 750, // 每 dpr 最大页面宽度

dpr = isIos ? Math.min(win.devicePixelRatio, 3) : 1,

scale = 1 / dpr,

tid;

docEl.removeAttribute('data-mw');

docEl.dataset.dpr = dpr;

metaEl = doc.createElement('meta');

metaEl.name = 'viewport';

metaEl.content = 'initial-scale=' + ratio + ',maximum-scale=' + ratio + ', minimum-scale=' + scale;

docEl.firstElementChild.appendChild(metaEl);

var refreshRem = function() {

var width = docEl.getBoundingClientRect().width;

if (width / dpr > maxwidth) {

width = maxwidth \* dpr;

}

var rem = width / 16;

docEl.style.fontSize = rem + 'px';

};

//...

refreshRem();

}

}

代码实现主要参考[淘宝网触屏版](http://m.taobao.com/)的适配方法，[查看效果和代码](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-rem.html)，其中 scss 的写法可以参见[这里](http://codepen.io/re54k/pen/aOoVLQ?editors=011)。

注意，较小的背景图（比如一些 icon）的 *background-size* 不要使用具体 rem 数值，裁剪后会出现边缘丢失。应使用与元素等尺寸切图，设定 background-size: contain|cover 来缩放。

**总结**

总的来看，目前还没有完美的解决方案，这些也只是尽可能的满足快速开发、简单适配需求的通用方案。其中对于一些较为细节的问题（比如字体的点阵尺寸、非弹性的定值需求）未做讨论。实际开发过程中，更应该综合考虑项目类型、资源成本、人员配合等多方面的因素，选择合适的方案。

代码实现中使用到的 **mobile-util.js** 对定宽和 rem 适配进行了整合，[源码在此](https://github.com/re54k/mobileweb-utilities/blob/master/util/mobile-util.js)。