概念

响应式网页设计最初是由 [Ethan Marcotte](http://alistapart.com/article/responsive-web-design) 提出的一个概念：为什么一定要为每个用户群各自打造一套设计和开发方案？Web设计应该做到根据不同设备环境自动响应及调整。当然响应式Web设计不仅仅是关于屏幕分辨率自适应以及自动缩放的图片等等，它更像是一种对于设计的全新思维模式；我们应当向下兼容、移动优先。

背景

**PC互联网加速向移动端迁移**：2012年12月底我国网民规模达到5.64亿，互联网普及率为42.1%，手机用户占网民总数的74.5%。预计到2015年，移动互联网的数据流量将超越PC端的流量。

**移动端入口**：当用户希望通过手机来完成PC页的操作时，常见的是商家的运营微博，期文案足够吸引用户点击链接参加活动，如果该活动页没做响应式处理：页面体积大、请求多、体验差、兼容性差，层层阻碍最终导致用户放弃参加。

优势

开发成本低，门槛低

* Native APP：Objective-C or Java – 学习成本高
* Hybrid APP： 外壳+Web APP，需安装。
* 响应式Web APP：HTML5+JS+CSS – 门槛低，极易上手，迭代快

跨平台和终端且不需要分配子域

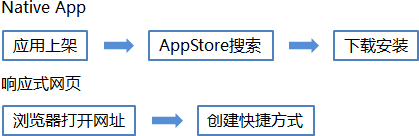
虽然可通过监测用户UA来判断用户终端后做跳转，但它还是分配了多个域，而响应式无需监测用户UA没有域的切换，只需根据终端类型来适配不同的功能模块与表现样式，它是跨平台和终端的，1页面适配多终端。

* PC – http://qzone.com
* Mobile – http://m.qzone.com
* 响应式：PC & Mobile – http://qzone.com 无需跳转

本地存储

Web App可以利用本地存储的特性将重要和重复的数据保存在本地，避免页面的重复刷新，减少重要信息在传输过程中被泄露，增量传输修改内容。

无需安装成本，迭代更新容易



更灵活、更方便的APP使用及安装方式将成为HTML5在移动平台上大放异彩的保障之一

实施

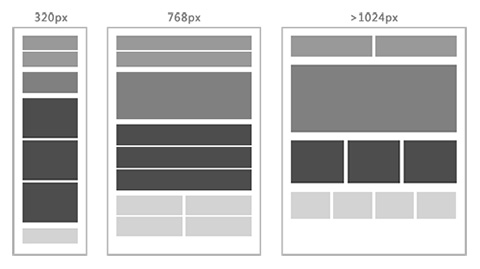
首先我们应该遵循**移动优先**原则，交互&设计应以移动端为主，PC则作为移动端的一个扩展；  
一个页面需要兼容不同终端，那么有两个关键点是我们需要去做到响应式的：

1. 响应式布局
2. 响应式内容（图片、多媒体）

**响应式布局**

如我们需要兼容不同屏幕分辨率、清晰度以及屏幕定向方式竖屏(portrait)、横屏(landscape)，怎样才能做到让一种设计方案满足所有情况？

那么我们的布局应该是一种弹性的栅格布局，不同尺寸下弹性适应，如以下页面中各模块在不同尺寸下的位置：



那么我们要怎么做？

Meta标签定义

使用 [viewport meta](http://www.quirksmode.org/mobile/viewports2.html) 标签在手机浏览器上控制布局

|  |  |
| --- | --- |
| 1 | <meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1" /> |

通过快捷方式打开时全屏显示

|  |  |
| --- | --- |
| 1 | <meta name="apple-mobile-web-app-capable" content="yes" /> |

隐藏状态栏

|  |  |
| --- | --- |
| 1 | <meta name="apple-mobile-web-app-status-bar-style" content="blank" /> |

iPhone会将看起来像电话号码的数字添加电话连接，应当关闭

|  |  |
| --- | --- |
| 1 | <meta name="format-detection" content="telephone=no" /> |

**神飞**：很多人常常使用initial-scale=1到非响应式网站上，这会让网站以100%宽度渲染而不会自动缩放，用户需要手动移动页面或者缩放。最差的是和initial-scale=1同时使用user-scalable=no或maximum-scale=1，这将使你的网站不能被缩放——用户不能放大/缩小网页来看到全部的内容。所以，请记住：如果你的网站不是响应式的，请不要使用initial-scale或者禁用缩放。

使用 [Media Queries](http://www.w3.org/TR/css3-mediaqueries/) 适配对应样式

常用于布局的CSS Media Queries有以下几种

设备类型(media type):

all所有设备  
screen 电脑显示器  
print打印用纸或打印预览视图  
handheld便携设备  
tv电视机类型的设备  
speech语意和音频盒成器  
braille盲人用点字法触觉回馈设备  
embossed盲文打印机  
projection各种投影设备  
tty使用固定密度字母栅格的媒介，比如电传打字机和终端

设备特性(media feature):

width浏览器宽度  
height浏览器高度  
device-width设备屏幕分辨率的宽度值  
device-height设备屏幕分辨率的高度值  
orientation浏览器窗口的方向纵向还是横向，当窗口的高度值大于等于宽度时该特性值为portrait，否则为landscape  
aspect-ratio比例值，浏览器的纵横比  
device-aspect-ratio比例值，屏幕的纵横比

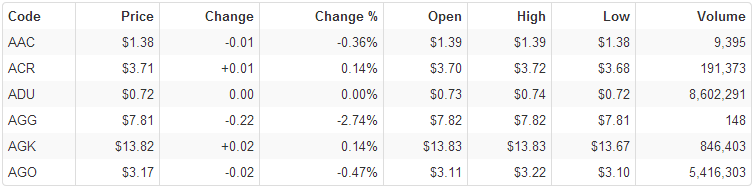
example:

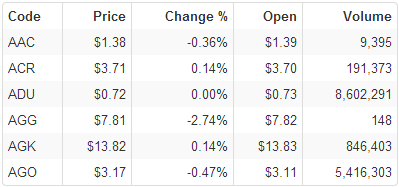
|  |  |
| --- | --- |
| 1  2  3  4 | */\* for 240 px width screen \*/*  @media only screen and (max-device-width:240px){  selector{ ... }  } |
| 1  2  3  4 | */\* for 320px width screen \*/*  @media only screen and (min-device-width:241px) and (max-device-width:320px){  selector{ ... }  } |

|  |  |
| --- | --- |
| 1  2  3  4 | */\* for 480 px width screen \*/*  @media only screen (min-device-width:321px)and (max-device-width:480px){  selector{ ... }  } |

适用于布局的 Media Queries 这里不做详述，可通过官方文档进一步了解  
那么对于表格(table)的响应式处理该是怎样的呢？我们该如何突破Table的局限性呢？  
接下来我们来了解以下的几种针对表格响应式处理的方法：

隐藏不重要数据列

  
处理前

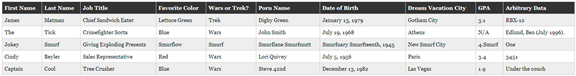
  
处理后

实现方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @media only screen and (max-width: 800px) {  table td:nth-child(2),  table th:nth-child(2) {**display**: none;}  }    @media only screen and (max-width: 640px) {  table td:nth-child(4),  table th:nth-child(4),  table td:nth-child(7),  table th:nth-child(7),  table td:nth-child(8),  th:nth-child(8){**display**: none;}  } |

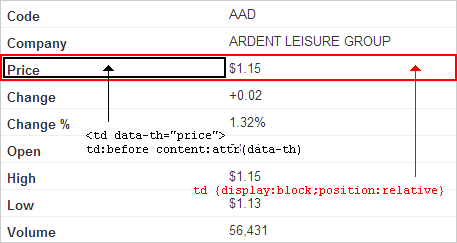
以用户角度思考，每个人对数据的认知不同，或许你隐藏的数据对于他却是很重要的。所以这种方法不推荐。

多列横向变2列纵向

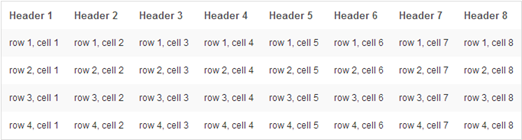
  
处理前

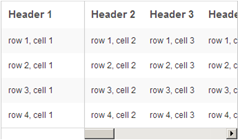
  
处理后

实现方法：<thead>定位隐藏，<td>变块元素，并绑定对应<th>列名，然后用伪元素的content:attr(data-th)实现<th>



固定首列，剩余列横向滚动

  
处理前

  
处理后

实现原理：

|  |  |
| --- | --- |
| 1  2 | thead {**float**:**left**;}  tbody {**display**:block;**width**:auto;overflow-x:auto;} |

栅格框架推荐

* [Responsive Grid System](http://responsive.gs/)
* [Fluid 960 Grid](http://www.designinfluences.com/fluid960gs)
* [Simple Grid](http://simplegrid.info/)

**响应式图片**



带宽是手机终端的硬伤，如果我们只是页面布局做了响应式处理，在我们用手机访问时，请求的图片还是PC上的大图；文件体积大，消耗流量多，请求延时长，因此导致的问题也是不可估量的。那么我们就得把图片也处理成响应式的根据终端类型尺寸分辨率来适配出合理的图形。

**处理原理：**浏览器获取用户终端的屏幕尺寸、分辨率逻辑处理后输出适应的图片，如屏幕分辨率320\*480，那么我们匹配给它的是宽度应小于320px的图片。如果终端屏幕的DPI(device pixels)[DPI详解](http://isux.tencent.com/mobile-development-essential-knowledge.html)值很高，也就是高清屏，那么我们就得输出2倍分辨率的图形(宽:640px)；以保证在高清屏下图形的清晰度。各种移动终端的屏幕参数可通过[http://screensiz.es/phone](http://screensiz.es/phone" \t "_blank)查询。

**解决方案：**其实W3C已经有一个用于响应式图形的草案：**新定义标签**[**<picture>**](https://dvcs.w3.org/hg/html-proposals/raw-file/9443de7ff65f/responsive-images/responsive-images.html)，因为它还只是草案，目前还没有支持的浏览器，期待在不久的未来我们能用上。虽然目前不支持，但我们还是来了解下，为之后的内容做个铺垫。

<picture>是一个图形element，内容由多个源图组成，并由CSS Media Queries来适配出合理图形，代码规范如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <picture width="500" height="500">  <source media="(min-width: 640px)" srcset="large-1.jpg 1x, large-2.jpg 2x">  <source media="(min-width: 320px)" srcset="med-1.jpg 1x, med-2.jpg 2x">  <source srcset="small-1.jpg 1x, small-2.jpg 2x">  <img src="small-1.jpg" alt="">  <p>Accessible text</p>  <!-- Fallback content-->  <noscript>  <img src="external/imgs/small.jpg" alt="Team photo">  </noscript>  </picture> |

source: 一个图片源；media: 媒体查询，用于适配屏幕尺寸；srcset: 图片链接，1x适应普通屏，2x适应高清屏；<noscript/>: 当浏览器不支持脚本时的一个替代方案；<img/>: 初始图片；另外还有一个无障碍文本，类似<img/>的alt属性。

<picture>目前还不支持，但它的原理我们是可借鉴的，所以就诞生了一个用于图片响应式处理的类库[Picturefill](http://github.com/scottjehl/picturefill" \t "_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <span data-picture data-alt="图片描述文本">  <span data-src="small.jpg"></span>  <span data-src="medium.jpg" data-media="(min-width: 400px)"></span>  <span data-src="large.jpg" data-media="(min-width: 800px)"></span>  <span data-src="extralarge.jpg" data-media="(min-width: 1000px)"></span>  <!-- 浏览器不支持JS时的备用方案. -->  <noscript>  <img src="external/imgs/small.jpg" alt="图片描述文本">  </noscript>  </span> |

其原理就是JS获取Source的源以及CSS Media Queries规则，输出适应图片， 逻辑细节这里不再解析，感兴趣的可查看其JS代码，逻辑不是很复杂，也可以自己封装一个类库，以适用于自身产品，例如图片加载失败的替代方案。

当然，在未来的 [CSS Image Level 4](http://dev.w3.org/csswg/css4-images/) 中已经实现了响应式图片的原生语法：[image-set](http://dev.w3.org/csswg/css4-images/" \l "image-set-notation" \t "_blank)

|  |  |
| --- | --- |
| 1  2 | <image-set> = image-set([ <image-set-decl>, ]\* [ <image-set-decl> | <color>])  <image-set-decl> = [ <image> | <string> ] <resolution> |

那么我们的响应式图片可以这样重写了

|  |  |
| --- | --- |
| 1  2 | **background-image**:url(*default.jpg*); */\* 普通幕 \*/*  **background-image**: -webkit-image-set(url(*medium.jpg*) 1x, url(*large.jpg*) 2x); */\* Retina高清屏 \*/* |

注：Webkit 目前只实现了 url() 形式的取值，且dppx值取负值[-2x]貌似也是合法的。

当然除此之外，还有其他的响应式处理，如服务端user-agent嗅探  
以下是部分项目地址，感兴趣的可以了解下：

* [Responsive Images JS Master Branch](https://github.com/filamentgroup/Responsive-Images)
* [Responsive images alt](https://github.com/filamentgroup/Responsive-Images)
* [Responsive Images and Context Aware Sizing](http://www.craig-russell.co.uk/responsive-images-and-context-aware-image-sizing/)
* [Responsive images with Doubletake.js](http://www.grahambird.co.uk/lab/doubletake/)
* [Responsive images with PHP and jQuery](http://www.jamesfairhurst.co.uk/posts/view/responsive_images_with_php_and_jquery/)
* [Responsive images using cookies](http://blog.keithclark.co.uk/responsive-images-using-cookies/)
* [Context aware responsive images](https://github.com/ahume/Responsive-Images)

**高分辨率(DPI)下的响应式处理**

1. SVG：优点可承载色彩丰富、设计复杂图形，且渲染不会出现边缘不顺滑；缺点是IE的支持不完美，在我大中华这是硬伤。
2. Icon fonts：支持多浏览器，图形颜色大小的修改成本低，易于维护；图形表现单一，不支持色彩丰富且复杂的图形，IE6渲染有毛边。
3. -webkit-image-set:只支持单个图形的适配，不利于图形合并，兼容不完美（Safari 6+, Chrome 21+）

JS检测：var retina = window.devicePixelRatio > 1;

CSS Media Query:

|  |  |
| --- | --- |
| 1  2  3  4 | @media (-webkit-min-device-pixel-ratio: 2), /\* Webkit-based browsers \*/  (min--moz-device-pixel-ratio: 2), */\* Older Firefox browsers (prior to Firefox 16) \*/*  (min-resolution: 2dppx), */\* The standard way \*/*  (min-resolution: 192dpi) */\* dppx fallback \*/* |

高DPI媒体查询规则将在下一篇文章中做详解，敬请期待

高分辨率下的1px border



由于高清屏的特性，1px是由2×2个像素点来渲染，那么我们样式上的border:1px在Retina屏下会渲染成2px的边框，与设计稿有出入，为了追求1px精准还原，我们就不得不拿出一个完美的解决方案

在Photoshop中，如果需要看似0.5px的边框，常见的方法就是对1px边框加上阴影模糊1px。同理，我们在retina屏下需要做到真实的1px边框，可利用box-shadow属性模拟。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | @media only screen and (-webkit-min-device-pixel-ratio:1.5),  only screen and (min-device-pixel-ratio:1.5) {    button {  **border**:none;  **padding**:0 16px;  box-shadow: inset 0 0 1px #000,  inset 0 1px 0 #75c2f8,  0 1px 1px -1px rgba(0, 0, 0, .5);  }    } |

[](http://isux.tencent.com/wp-content/uploads/2013/07/20130712101225160.html)

留给我们的思考

* 响应式不只是技术的实现，它更像是一种对于设计的全新思维模式
* 浏览的体验短期内还无法超越原生应用
* 左手操作习惯的交互
* Webapp的消息推送
* 调用本地文件系统的能力弱
* 响应式图片的解决方案
* 对PC事件的兼容
* WebAPP页面体积的响应式适配
* 代码实现和内容可维护性之间的权衡
* 控制设计开发成本

实用技巧

* 点击区域不限于元素的视觉区域，便于用户点击。同时排版不受限制，可以达到原生App的视觉效果。
* 气泡框可以减少页面跳转，适合消息提醒等微任务的处理。
* 信息架构上越来越接近原生App，有利于扁平化层级关系和减少界面跳转等设计元素将得到更多的应用。
* 识别更多的手势操作，如下拉刷新和右滑存档等平移手势。操作不必全部呈现在界面中，和平台操作保证一致。
* 调用系统硬件，如重力感应等传感器、多媒体设备，不过在手机端还鲜有应用案例，离大规模应用还有一定的距离。