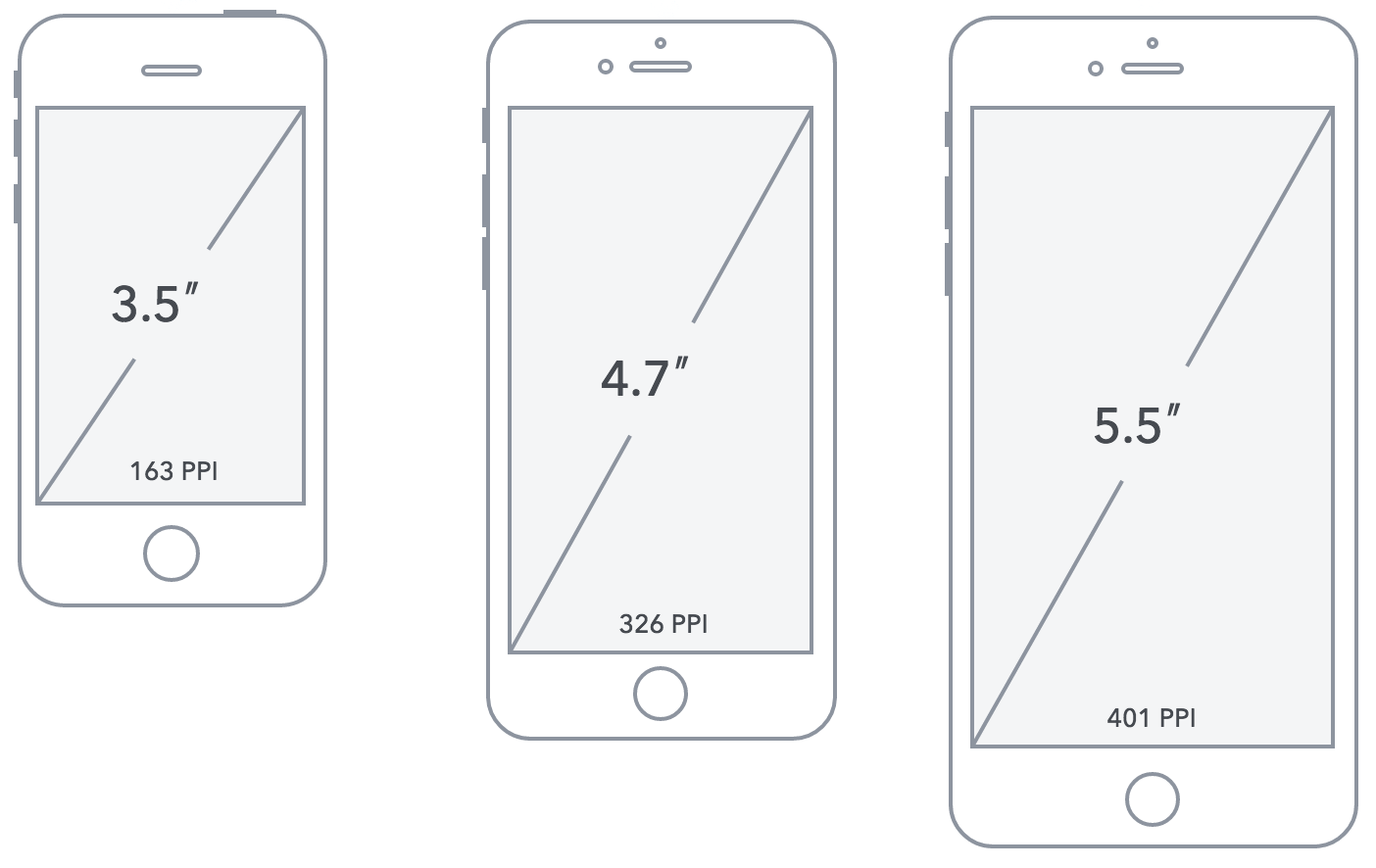
**移动Web**

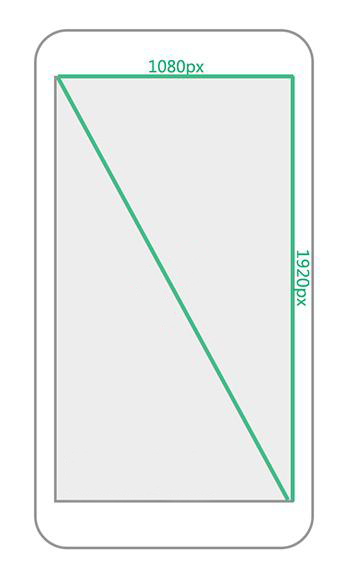
# 1 基础知识

## 1.1 屏幕

移动设备与PC设备最大的差异在于屏幕，这主要体现在屏幕尺寸和屏幕分辨率两个方面。通常我们所指的屏幕尺，实际上指的是屏幕对角线的长度（一般用英寸来度量）如下图所示：

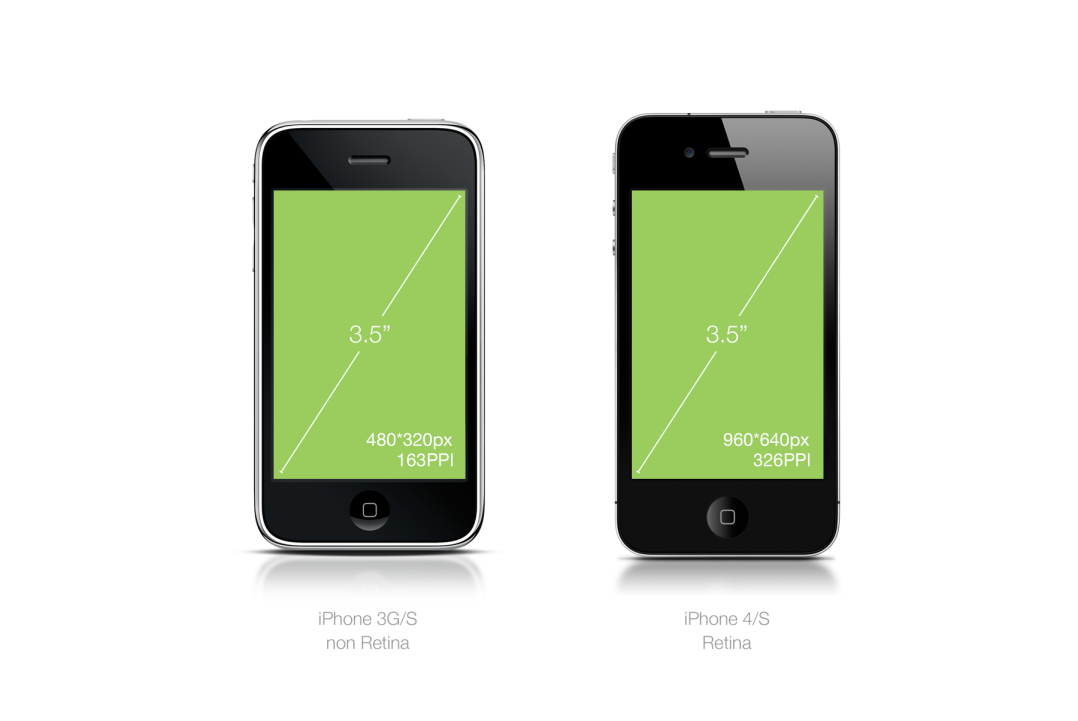


分辨率则一般用像素来度量，表示屏幕水平和垂直方向的像素数，例如1920\*1080指的是屏幕垂直方向和水平方向分别有1920和1080个像素点而构成，如下图所示



## 1.2 长度单位

在Web开发中可以使用px（像素）、em、pt（点）、in（英寸）、cm（厘米）做为长度单位，我们最常用px（像素）做为长度单位。我们可以将上述的几种长度单位划分成相对长度单位和绝对长度单位。

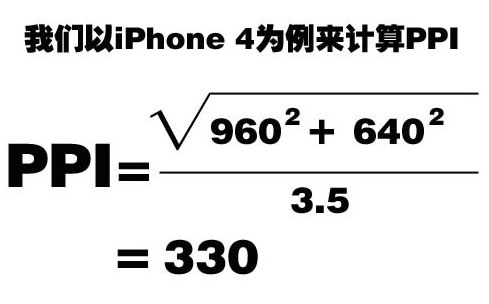


如上图所示，iPhone3G/S和iPhone4/S的屏幕尺寸都为3.5英寸（in）但是屏幕分辨率却分别为480\*320px、960\*480px，由此我们可以得出英寸是一个绝对长度单位，而像素是一个相对长度单位（像素并没有固定的长度）。

## 1.3 像素密度

**DPI（Dots Per Inch）**是印刷行业中用来表示打印机每英寸可以喷的墨汁点数，计算机显示设备从打印机中借鉴了DPI的概念，由于计算机显示设备中的最小单位不是墨汁点而是像素，所以用**PPI（Pixels Per Inch）**值来表示屏幕每英寸的像素数量，我们将PPI、DPI都称为**像素密度**，但PPI应用更广泛，DPI在Android设备比较常见。

如下图所示，利用 *勾股定理* 我们可以计算得出PPI



PPI值的越大说明单位尺寸里所能容纳的像素数量就越多，所能展现画面的品质也就越精细，反之就越粗糙。

**Retina**即视网膜屏幕，苹果注册的命名方式，意指具有较高PPI（大于320）的屏幕。

思考：在屏幕尺寸（英寸）固定时，PPI和像素大小的关系？

结论：当PPI 越大，像素的实际大小就会越小，当PPI越小，像素实际大小就越大。

retina：一种具备超高像素密度的液晶屏，同样大小的屏幕上显示的像素点由1个变为多个，如在同样带下的屏幕上，苹果设备的retina显示屏中，像素点1个变为4个

在高清显示屏中的位图被放大，图片会变得模糊，**因此移动端的视觉稿通常会设计为传统PC的2倍**。

那么，前端的应对方案是：

设计稿切出来的图片长宽保证为偶数，并使用backgroud-size把图片缩小为原来的1/2

//例如图片宽高为：200px\*200px，那么写法如下

.css{width:100px;height:100px;background-size:100px 100px;}

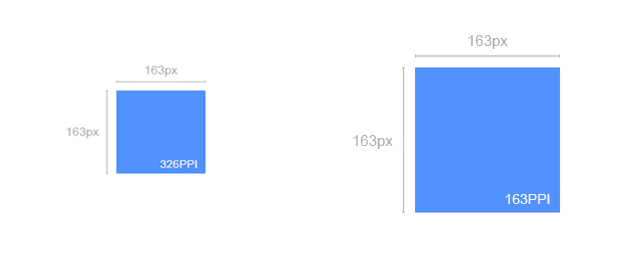
其它元素的取值为原来的1/2，例如视觉稿40px的字体，使用样式的写法为20px

.css{font-size:20px}

## 1.4 设备独立像素

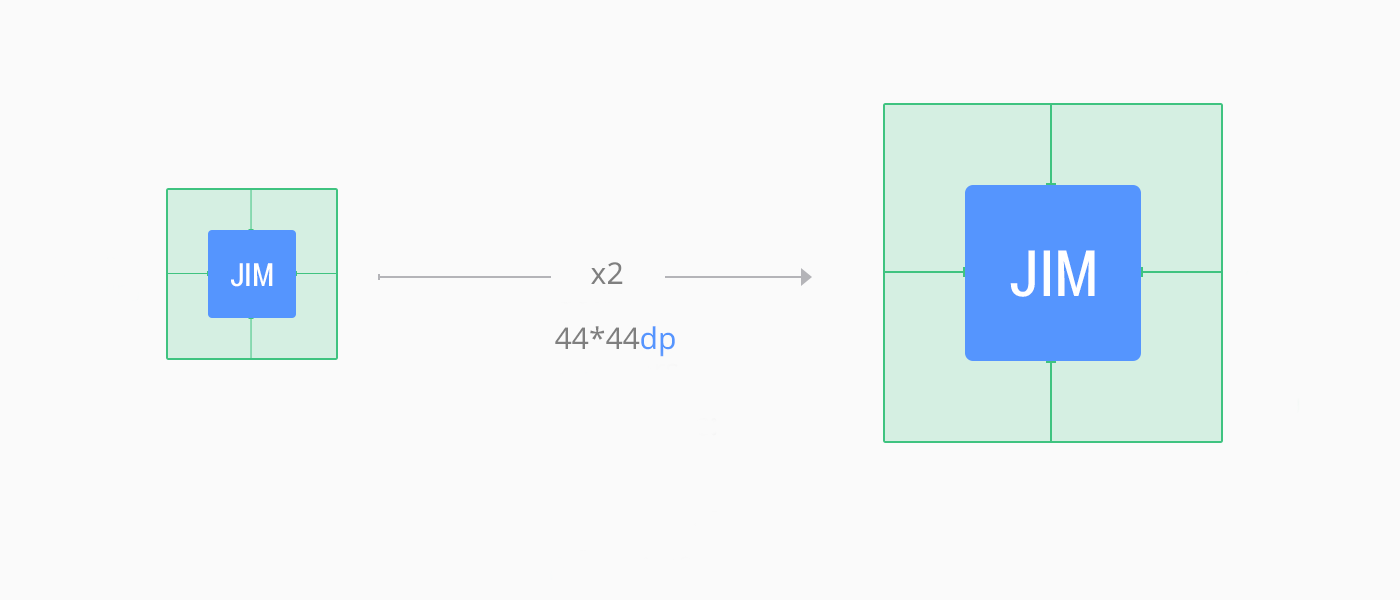
随着技术发展，设备不断更新，出现了不同PPI的屏幕共存的状态（iPhone3G/S为163，iPhone4/S为326），这时像素不再是统一的度量单位，会造成同样尺寸的图像在不同PPI设备上的显示大小不一样。

如下图，假设你设计了一个163\*163的蓝色方块，在PPI为163的屏幕上，那这个方块看起来正好就是1\*1寸大小，在PPI为326的屏幕上，这个方块看起来就只有0.5\*0.5寸大小了。



做为用户是不会关心这些细节的，他们只是希望在不同PPI的设备上看到的图像内容差不多大小，所以这时我们需要一个**新的单位**，这个新的单位能够保证图像内容在不同的PPI设备看上去大小应该差不多，这就是独立像素，IOS设备上叫PT(Point)，Android设备上叫DIP(Device independent Pixel)或DP。

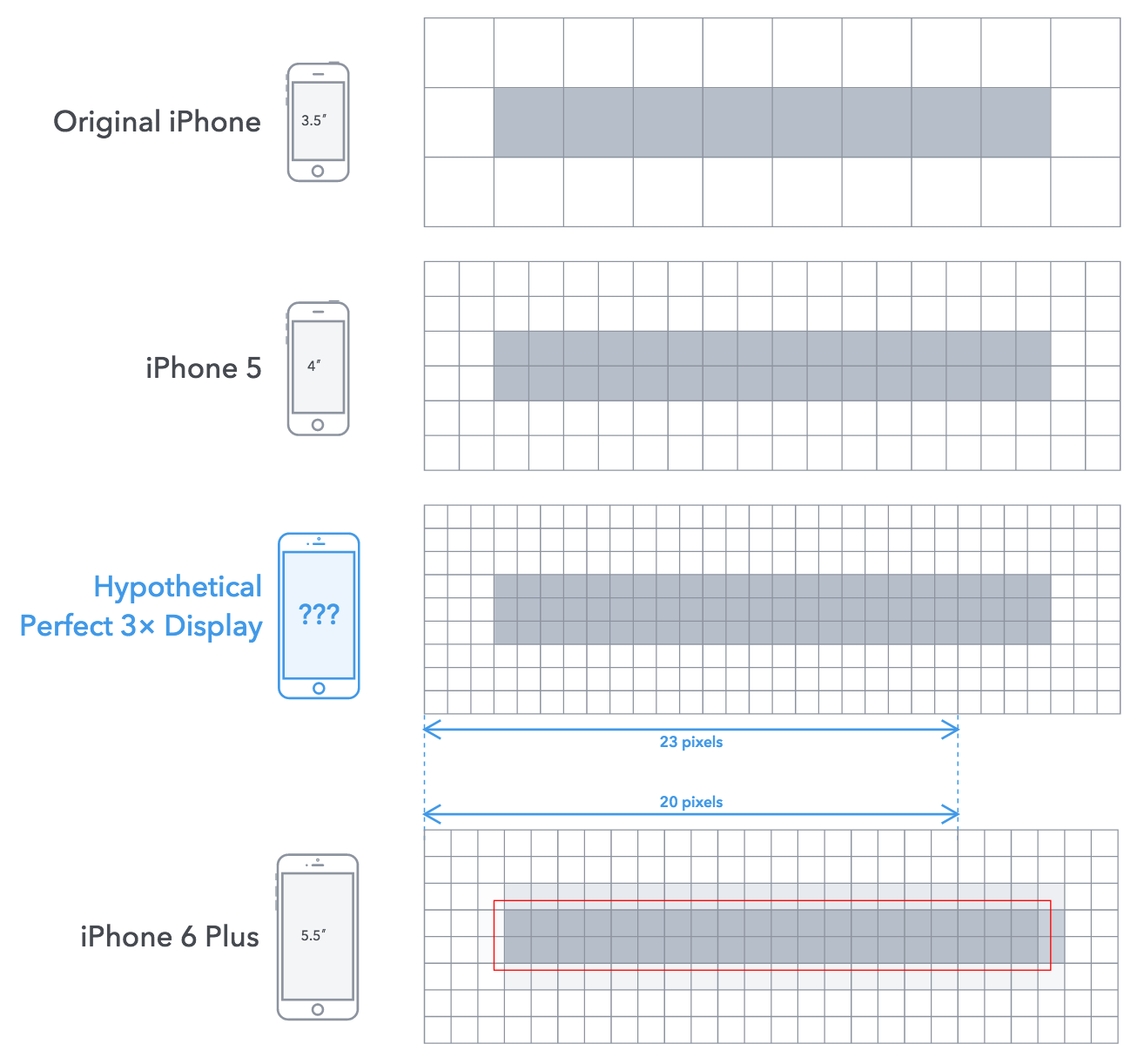
举例说明就是iPhone 3G（PPI为163）1dp = 1px，iPhone 4（PPI为326）1dp = 2px。



我们也不难发现，如果想要iPhone 3G/S和iPhone 4/S图像内容显示一致，可以把iPhone 4/S的尺寸放大一倍（它们是一个2倍(@2x)的关系），即在iPhone3G/S的上尺寸为44\*44px，在iPhone4/S上为88\*88px，我们要想实现这样的结果可以设置44\*44dp ,这时在iPhone3G/S上代表44\*44px，在iPhone4/S上代表88\*88px。

通过上面例子我们不难发现dp同px是有一个对应（比例）关系的，这个对应（比例）关系是操作系统确定并处理，通过window.devicePixelRatio可以获得该比例值。

下图展示了iPhone不同型号间dp和px的比例关系



从上图我们得知dp（或pt）和px并不总是绝对的倍数关系，而是window.devicePixelRatio ~= 物理像素/独立像素，然而这其中的细节不必关心，操作系统会自动处理好（保证1dp在不同设备上看上去大小差不多）。

## 1.5 像素

物理像素指的是屏幕渲染图像的最小单位，不可人为进行改变，其值大小决定了屏幕渲染图像的品质，以上所讨论的都指物理像素。

// 获取屏幕的物理像素尺寸

window.screen.width;

window.screen.height;

CSS像素指的是通过CSS进行网页布局时用到的单位，其默认值(PC端)是和物理像素保持一致的（1个单位的CSS像素等于1个单位的物理像素），但是我们可通缩放改变其大小。

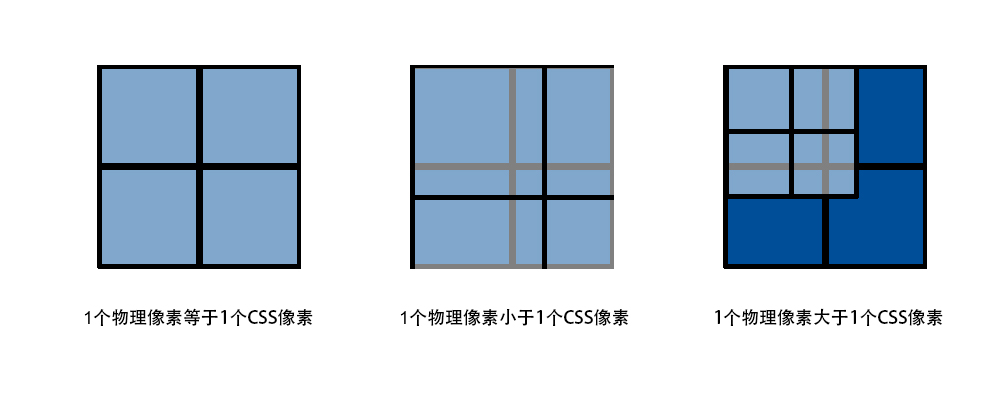
浏览器窗口尺寸是CSS像素来度量的，部分浏览器存在兼容问题（IE低版本无此属性），但是可以忽略这些细节。

// 获取浏览器窗口尺寸 此值大小包含滚动条的大小

window.innerWidth;

window.innerHeight;

我们通过调整浏览器的缩放比例可以直观的理解CSS像素与物理像素之前的对应关系，如下图所示：



我们需要理解的是物理像素和CSS像素的一个关系，1个物理像素并不总是等于一个CSS像素，通过调整浏览器缩放比例，可以有以上3种情况。

# 2 视口

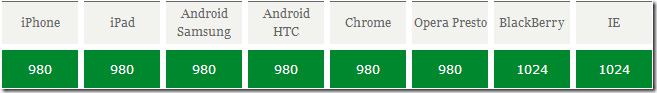
## 2.1 视口的产生

视口（**viewport**）是用来约束你网站中最顶级块元素<html>的，即它决定了<html>的大小。

我们知道移动设备屏幕普遍都是比较小的，但是大部分的网站又都是为PC设备来设计的，要想让移动设备也可以正常显示网页，移动设备不得不做一些处理，只要viewport足够大（块级元素html就会比较大），就能保证原本为PC设备设计的网页也能在移动设备上正常显示，移动设备厂商也的确是这样来处理的。

在移动设备上viewport不再受限于浏览器的窗口，而是允许开发人员自由设置viewport的大小，通常浏览器会设置一个默认大小的viewport，同时为了能够正常显示那些专为PC设计的网页，一般这个值的大小会大于屏幕的尺寸。

如下图为常见默认viewport大小（仅供参考）



从图中统计数我们得知不同的移动厂商分别设置了一个默认的viewport的值，这个值保证大部分网页可以正常在移动设备下浏览。

## 2.2 产生缩放和滚动条的原因

移动设备上有2个viewport，分别是layout viewport和ideal viewport。

1、layout viewport（布局视口）指的是我们可以进行网页布局区域的大小，同样是以CSS像素做为计量单位，可以通过下面方式获取

// 获取layout viewport

document.documentElement.clientWidth;

document.documentElement.clientHeight;

通过前面介绍我们知道，如果要保证为PC设计的网页在移动设备上布局不发生错乱，移动设备会默认设置一个较大的viewport（如IOS为980px），这个viewport实际指的是layout viewport，我们也可改变这个默认值，后面将会介绍。

2、ideal viewport（理想视口）设备屏幕区域，以dp（或pt、px）做为计量单位，不同的设备具有不同大小的理想视口，其大小是不可能被改变的，通过下面方式可以获取。

// 获取ideal viewport有两种情形

// 新设备

window.screen.width;

window.screen.height;

// 老设备

window.screen.width / window.devicePixelRatio;

window.screen.height / window.devicePixelRatio;

在理解了上述三个viewport后我们来解释为什么网页会被缩放或出现水平滚动条，其原因在于移动设备浏览器会默认设置一个layout viewport，并且这个值会大于ideal viewport，那么我们也知道ideal viewport就是屏幕区域，layout viewport是我们布局网页的区域，那么最终layout viewport最终是要显示在ideal viewport里的，而layout viewport大于ideal viewport时，于是就出现滚动条了，那么为什么有的移动设备网页内容被缩放了呢？移动设备厂商认为将网页完整显示给用户才最合理，而不该出现滚动条，所以就将layout viewport进行了缩放，使其恰好完整显示在ideal viewport（屏幕）里，其缩放比例为ideal viewport / layout viewport。

## 2.3 移动浏览器

移动端开发主要是针对IOS和Android两个操作系统平台的，除此之外还有Windows Phone。

移动端主要可以分成三大类，系统自带浏览器、应用内置浏览器、第三方浏览器

系统浏览器：指跟随移动设备操作系统一起安装的浏览器。

应用内置浏览器：通常在移动设备上都会安装一些APP例如QQ、微信、微博、淘宝等，这些APP里往往会内置一个浏览器，我们称这个浏览器为应用内置浏览器（也叫WebView），这个内置的浏览器一般功能比较简单，并且客户端开发人员可以更改这个浏览器的某些设置，在我们理实的开发里这个浏览器很重要。

第三方浏览器：指安装在手机的浏览器如FireFox、Chrome、360等等。

在IOS和Android操作系统上自带浏览器、应用内置浏览器都是基础Webkit内核的。

# 3 屏幕适配

经过分析我们得到，移动页面最理想的状态是，避免滚动条且不被默认缩放处理，我们可以通过设置<meta name="viewport" content="">来进行控制，并改变浏览器默认的layout viewport的宽度。

## 3.1 Viewport详解

viewport 是由苹果公司为了解决移动设备浏览器渲染页面而提出的解决方案，后来被其它移动设备厂商采纳，其使用参数如下：

// 通过设置属性content=""实现，中间以逗号分隔

// 例如 <meta name="viewport" content="width=device-width">

width 设置layout viewport 宽度，其取值可为数值或者 device-width。

height 设置layout viewport 高度，其取值可为数值或者device-height

initial-scale设置页面的初始缩放值，为一个数字，可以带小数。

maximum-scale允许用户的最大缩放值，为一个数字，可以带小数。

minimum-scale允许用户的最小缩放值，为一个数字，可以带小数。

user-scalable是否允许用户进行缩放，值为"no"或"yes"。

注：device-width 和 device-height实为ideal viewport的宽高

## 3.2控制缩放

1、设置<meta name="viewport" content="initial-scale=1">，这时我们发现网页没有被浏览器设置缩放。

2、设置<meta name="viewport" content="width=device-width">，这时我们发现网页也没有被浏览器设设置缩放。

当我们设置width=device-width，也达到了initial-scale=1的效果，得知其实 initial-scale = ideal viewport / layout viewport。

两种方式都可以控制缩放，开发中一般同时设置（为了解决一些兼容问题），即<meta name="viewport" content="width=device-width, initial-scale=1.0">

## 3.3 避免滚动

我们知道，滚动条是 layout viewport 相对于 ideal viewport 的，所以只要设置：

 layout viewport 小于或等于 ideal viewport，

即<meta name="viewport" content="width=device-width">。

经测试发现我们并没有完全的解决滚动条的问题，原因在于我们示例里的.box {width: 1024px;}设置了一个绝对的宽度造成的，要解决这个问题我们可以设置一个百分比（100%）的宽度。

## 3.4 百分比适配

1、设置<meta name="viewport" content="width=device-width, initial-scale=1">

2、设置页面宽度为百分比

**关于百分比**

我们需要重新认识CSS里百分比的使用，

// 测试下列属性设置为百分比

width 参照父元素的宽度

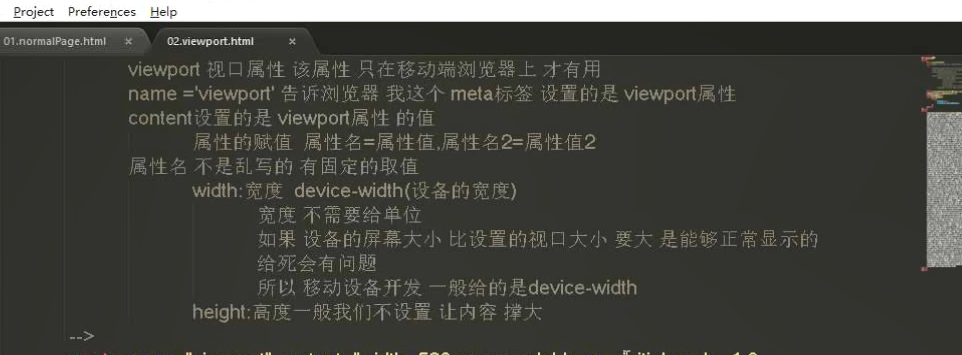
height 参照父元素的高度

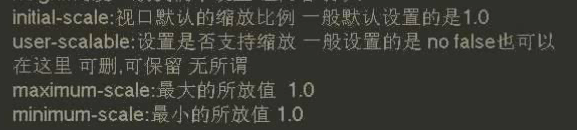
padding 参照父元素的宽度

border 不支持百分比设置

margin 参照父元素的宽度

## 3.5 rem适配





1、设置<meta name="viewport" content="width=device-width, initial-scale=1">

2. 设置页面元素宽度单位为 rem 或 em

**关于em和rem**

em 相对长度单位，其参照当前元素字号大小，如果当前元素未设置字号则会继承其祖先元素字号大小 例如

.box {font-size: 16px;} 则 1em = 16px

.box {font-size: 32px;} 则 1em = 32px；

rem 相对长度单位，其参照根元素(html)字号大小 例如

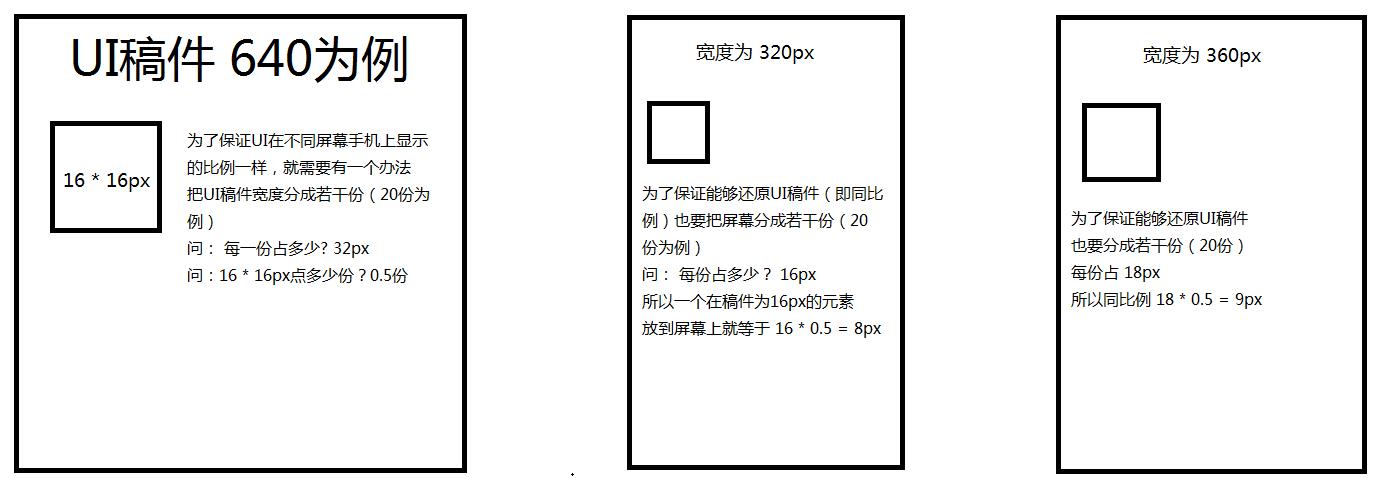
html {font-size: 16px;} 则 1rem = 16px

注：所有浏览器默认字号都是16px，所以即使没有书写字体大小，直接给某些元素按照百分比设置文字大小也是正确的。

经常会有字体大小设置为：62.5%，然后后续设置问大小写为

1em 就是10px 1.2em就是12px

（某些安卓手机可以调置系统字号后，浏览器默认字号会受影响）



# 4 触屏事件



## 4.1 事件类型

touchstart: 手指触摸屏幕时触发

touchmove: 手指在屏幕上移动时触发

touchend: 手指离开屏幕时触发

## 4.2 TouchEvent对象

touches: 位于屏幕上的所有手指的列表

targetTouches: 位于该元素上的所有手指的列表

changedTouches：touchstart时包含刚与触摸屏接触的触点，touchend时包含离开触摸屏的触点

注：没有对比出touches同targetTouches的差异，推荐使用targetTouches

## 4.3 Touch对象

clientX/Y 手指相对 layout viewport的水平/垂直像素距离

pageX/Y 手指相对于layout viewport的水平/垂直像素距离（包含滚动）

screenX/Y 手指相对于layout viewport的水平/垂直像素距离（包含滚动）

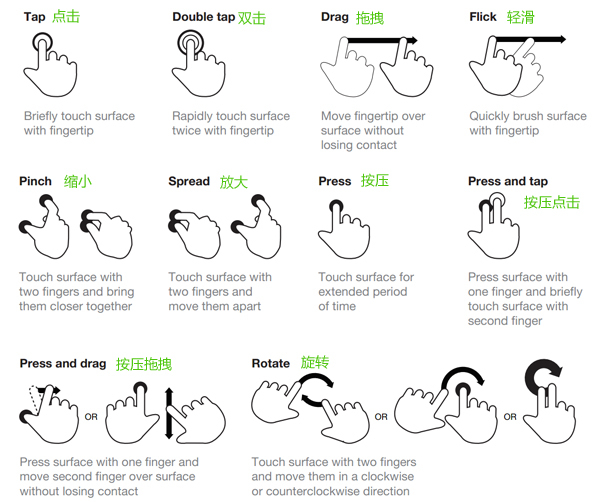
target 手指最初与屏幕接触时的元素

移动开发通常会设置<meta name="viewport" content="width=device-width, initial-scale=1">，这时这三对坐标值是完全一样的。

## 4.4 click延时

早期移动设备浏览器网页时内容非常小，为了增强用户体验，苹果公司专门为移动设备设计了双击放大的功能，确保用户可以非常方便的放大网页内容，但是当用户单击一个按钮时，移动设备会延时（约300ms）执行，判断用单是否要双击。用触屏事件可以解决这个问题。

## 4.5 手势封装



利用触屏事件简易封装手势，主要用途是熟悉触屏事件的使用

1、tap检测接触和离开屏幕的距离来实现

2、drag跟踪手指移动位置，进而设置元素定位坐标

3、swipe 判断手指滑动的方向

# 5 zepto.js

zeptojs为我们封装了常的触屏事件，需要touch模块支持，默认没有构建此模，我们可以自定义构建。

1、安装Nodejs环境

2、下载[zepto.js](https://github.com/madrobby/zepto)

3、解压缩

4、cmd命令行进入解压缩后的目录

5、执行npm install 命令

6、编辑make文件，添加自定义模块并保存，如下图



7、然后执行命令 npm run-script dist

8、查看目录dist即构建好的zepto.js

# 6 调试工具

## 6.1 模拟调试

现代主流浏览器均支持移动开发模拟调试，通常按F12可以调起，其使用也比较简单，可以帮我们方便快捷定位问题。

## 6.2 真机调试

模拟调试可以满足大部分的开发调试任务，但是由于移动设备种类繁多，环境也十分复杂，模拟调试容易出现差错，所以真机调试变的非常必要。

有两种方法可以实现真机调试：

1、将做好的网页上传至服务器或者本地搭建服务器，然后移动设备通过网络来访问。

2、借助第三方的调试工具，如weinre、debuggap、ghostlab等

真机调试必须保证移动设备同服务器间的网络是相通的。

# 7 媒体查询

设备终端的多样化，直接导致了网页的运行环境变的越来越复杂，为了能够保证我们的网页可以适应多个终端，不得不专门为某些特定的设备设计不同的展示风格，要实现这个目标的前提是必须有能力了解当前网页是运行什么终端设备，通过媒体查询可以做到这一点。

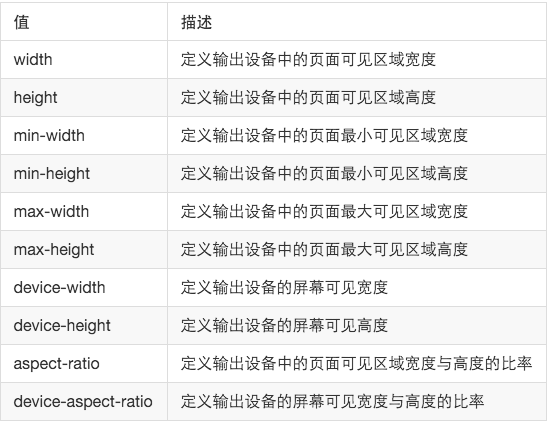
## 7.1 媒体类型

可以通过媒体类型对不同的设备指定不同的样式，一般我们使用screen。



## 7.2 媒体特性

判断条件或者查询依据

****

## 7.3 and not only

1、and 可以将多个媒体特性连接到一起，相当于“且”的意思。

2、not 排除某个媒体类型，相当于“非”的意思，可以省略。

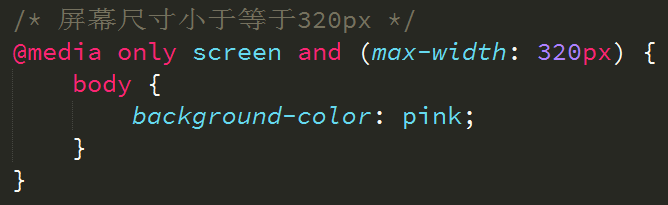
3、only指定某个特定的媒体类型，可以省略。

## 7.4 引入方式

1、link方法，见代码示例6-1.html

<link href="./6-1.css" media="only screen and (max-width: 320px)">

2、@media方法（写在CSS里）



## 7.5 常用特性

1、width / height 完全等于layout viewport

2、max-width / max-height 小于等于layout viewport

3、min-width / min-height 大于等于layout viewport

4、device-width / device-height 完全等于ideal viewport

5、orientation: portrait | landscape 肖像/全景模式

# 8 CSS预处理器-Less

CSS 预处理器是一种语言，用来为 CSS 增加一些编程的的特性，无需考虑浏览器的兼容性问题，并且你可以在 CSS 中使用变量、简单的程序逻辑、函数等等在编程语言中的一些基本技巧，可以让你的 CSS 更简洁，适应性更强，代码更直观等诸多好处。

常见的CSS预处理器有：LESS、SASS、Stylus等

安装

1、安装Nodejs环境 Node Package Manager (验证 node -v npm -v)

2、打开控制台（cmd），执行npm install -g less (验证 lessc -v)

3、执行 npm install -g less-plugin-clean-css

4、命令行编译 lessc path/xxx.less path/xxx.css

编译

一般都会借助于编辑器插件来完成编译任务，以sublime\_text为例

1、ctrl+shit+p打开命令面板

2、输入install package然后回车

3、安装 LESS、lessc、Less2Css三个插件

4、alt+s快捷键即可实现编译

语法

1、变量

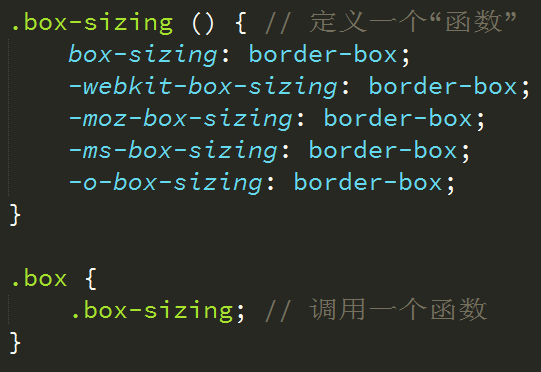
格式：@变量名: 值，定义完成后可以重复使用



见代码示例7-1.less

2、混合

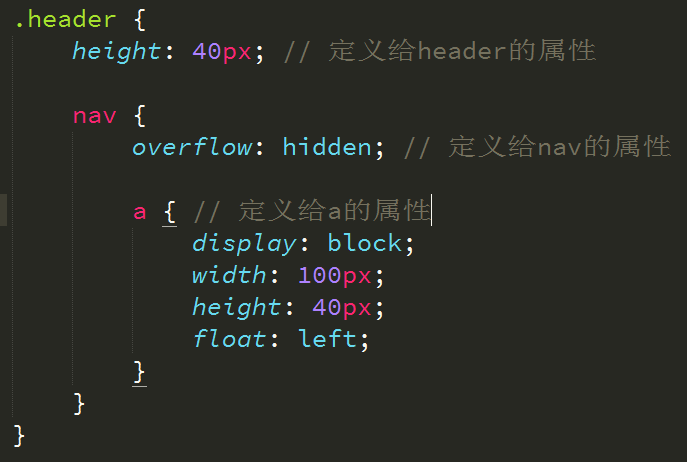
我们可以像使用函数一样来使用CSS



见代码示例7-2.less

3、嵌套

嵌套可以非常方便的管理我们的CSS层级关系



浏览器中使用

我们可以引入一个less.js文件，实现实时的解析，而不必每次修改都要编译，最后完成所有开发任务后，再通过编辑器编译成css文件。

1、[下载](https://raw.github.com/less/less.js/v2.5.3/dist/less.min.js)然后引入less.js

2、引入xx.less文件，如：

<link rel="stylesheet/less" type="text/css" href="styles.less" />

注意：rel属性必须指定成stylesheet/less，并且styles.less要先于less.js引入。

必须以服务器方式访问，可以放到study目录下，或者webstrom自带服务器功能，或者我们的 ghostlab调试工具的服务器。

# 9 移动端类库



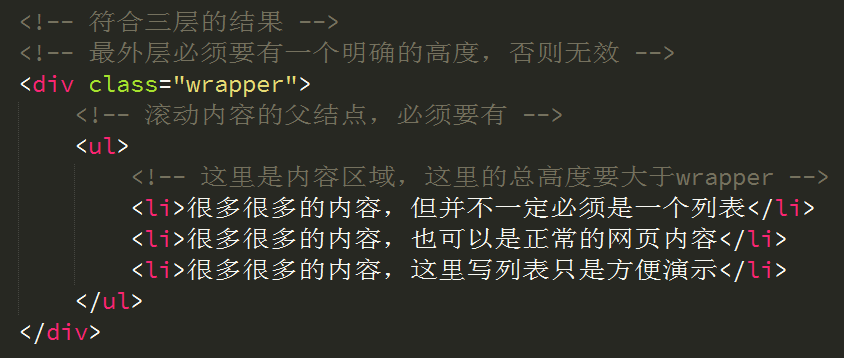
### 9.1 iScroll.js

一个可以实现客户端原生滚动效果的类库。

1、[下载iScroll](https://github.com/cubiq/iscroll)

2、build目录下提供了不同版本的iScroll，可根据情况选择使用

3、html要求有3层结构如下图



4、获取wrapper这个最外层结点，然后实例化，如下图

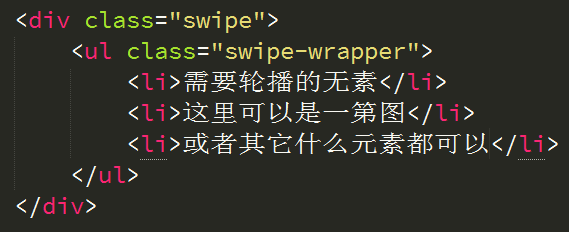


见示例代码8-1.html

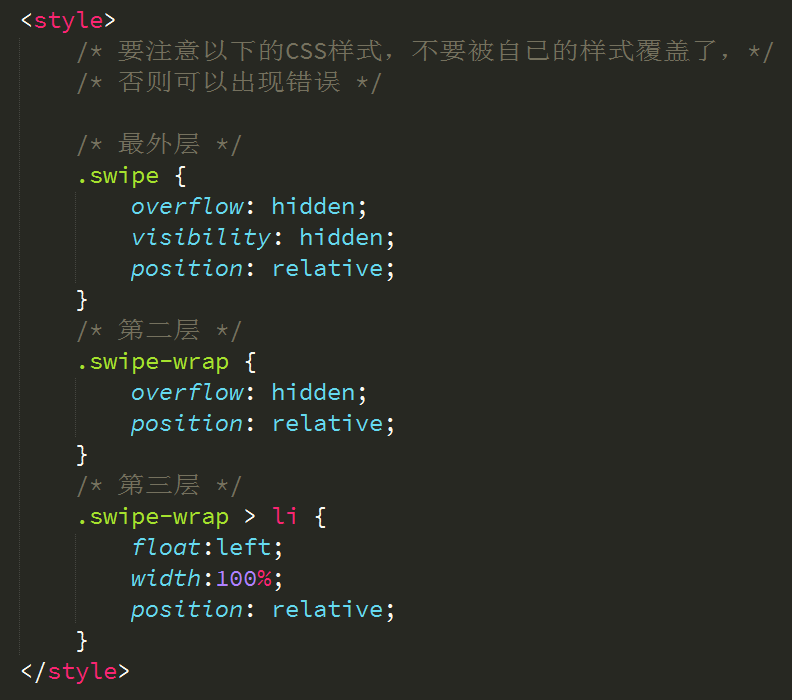
### 9.2 swipe.js

1、[下载swipe.js](https://github.com/thebird/Swipe)

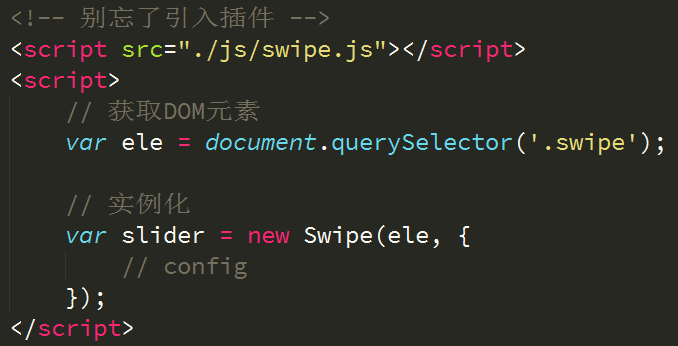
2、html结构要求有三层结构，如下图



4、需要一些基础CSS样式，这些样式要对应到html结构上，如下图



5、获取swipe元素，然后实例化，如下图



### 9.3 swiper.js

1、[下载swiper.js](https://github.com/nolimits4web/Swiper/)，

2、其[中文网站](http://www.swiper.com.cn/)非常详细介绍了其使用方法

### 9.4 fastclick.js

在移动设备上为了提升click的响应速度，我们选择了使用Zepto事件封装的tap来进行模拟，但是这会带来一个副作用，这个副作用就是“点透”，我们通过一个例子来解释“点透”，见示例代码8-3.html（自行查阅点透现象发生的原因）

从上可以看出Zepto.js有不完善的地方，并且我们有时也希望我们的移动版页面在PC端上也可用，但是PC端是不支持touch事件的，这时我们面临的问题是即提升click在移动设备上的响应速度，又不能使用Zepto.js的tap事件，这时fastclick可以解决这个问题。



如图实际应用的场景，当点击半闭按钮时，如果下面有click事件或链接则会被触发。

1、[下载fastclick](https://github.com/ftlabs/fastclick)

2、引入lib目录下的fastclick.js

2、设用方法即可，如下图



# 10网页布局

## 10.1 布局方式

1、固定宽度布局：为网页设置一个固定的宽度，通常以px做为长度单位，常见于PC端网页。

2、流式布局：为网页设置一个相对的宽度，通常以百分比做为长度单位。

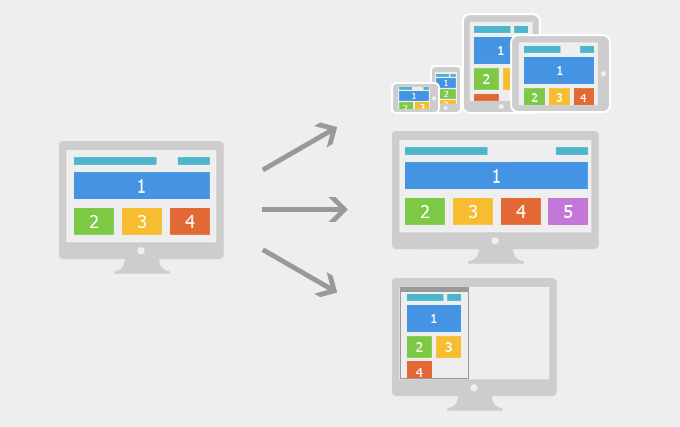
3、栅格化布局：将网页宽度人为的划分成均等的长度，然后排版布局时则以这些均等的长度做为度量单位，通常利用百分比做为长度单位来划分成均等的长度。

4、响应式布局：通过检测设备信息，决定网页布局方式，即用户如果采用不同的设备访问同一个网页，有可能会看到不一样的内容，一般情况下是检测设备屏幕的宽度来实现。

注：以上几种布局方式并不是独立存在的，实际开发过程中往往是相互结合使和的。

## 10.2 响应式布局

Responsive design，意在实现不同屏幕分辨率的终端上浏览网页的不同展示方式。通过响应式设计能使网站在手机和平板电脑上有更好的浏览阅读体验。



如上图所示，屏幕尺寸不一样展示给用户的网页内容也不一样，我们利用媒体查询可以检测到屏幕的尺寸（主要检测宽度），并设置不同的CSS样式，就可以实现响应式的布局。

我们利用响应式布局可以满足不同尺寸的终端设备非常完美的展现网页内容，使得用户体验得到了很大的提升，但是为了实现这一目的我们不得不利用媒体查询写很多冗余的代码，使整体网页的体积变大，应用在移动设备上就会带来严重的性能问题。

响应式布局常用于企业的官网、博客、新闻资讯类型网站，这些网站以浏览内容为主，没有复杂的交互。

一般我们会对常见的设备尺寸进行划分后，再分别确定为不同的尺寸的设备设计专门的布局方式，如下图所示



以上是我们对常见的尺寸进行分类后的结果，下图是与之对应的媒体查询条件。



**有最大、最小宽度的百分比自适应布局**

适用场景：门户网站首页，图片较多的首页。

**百分比自适应布局**

适用场景：信息文字较多的网页，内容较多网页。

**全屏自适应布局**

适用场景：单页面网页，移动web app 页面。

# 11 CSS框架

随着Web应用变的越来越复杂，在大量的开发过程中我们发现有许多功能模块非常相似，比如轮播图、分页、选项卡、导航栏等，开发中往往会把这些具有通用性的功能模块进行一系列封装，使之成为一个个组件应用到项目中，可以极大的节约开发成本，将这些通用的组件缩合到一起就形成了前端框架。

常见的CSS框架有

Bootstrap：

Amaze UI：国人开发，后起之秀！

Framework7：开源免费的框架可以用来开发混合移动应用（原生和HTML混合）或者开发 iOS & Android 风格的WEB APP。