# \*\*\*\*\*\*移动端\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1.相关概念

1、屏幕尺寸

对角线的长度 英寸表示 1英寸 ≈ 2.54cm

2、屏幕分辨率（iphone6 750\*1334）

横纵向上像素点数（物理像素点）

3、屏幕像素密度

单位英寸上所包含的物理像素点数

这个值求法： 与屏幕尺寸和屏幕分辨率关系密切

通过分辨率和勾股定理可以求出对角线上总的像素点数

4、物理像素 （物理硬件描述）

设备固定的,出厂时就确定了

5、css像素 （用户描述）

抽象的 用来精确的度量web页面的尺寸大小的 给web开发人员使用

通常我们在css以及js当中所使用的都是css像素；

6、设备独立像素 （计算机系统描述，包含css像素）

设备独立像素是为了让物理像素更好的展现用户层的像素而出现的，是为了转化；

在某些特定的特定的条件下，css像素可以 = 设备独立像素；

也是抽象的虚拟像素

7、像素比（dpr）

物理像素和设备独立像素之间的比例；（可以让设备独立像素和物理像素拉关系）

普通屏：不考虑任何的缩放 1css像素 = 1物理像素 = 1设备独立像素

高清屏 dpr = 2： 1 2 1

dpr = 3 1 3 1

8、思考：

在不考虑任何缩放的前提下：（前提是它）

物理像素和css像素的关系是什么？

要根据dpr去计算

css像素和设备独立像素的关系是什么？

css像素 = 设备独立像素

9、最终屏幕宽度在理想完美状态用各种像素的描述各是多少？

物理像素 750px

设备独立像素 375px

css像素 375px

10、pc视口

四种求宽高的方法

11、布局视口

为了转移pc的网页，而引入的一个容器。它的大小是要远大于设备独立像素，各个设备差异不大。（980）

为了把这个布局视口当中的元素全部展现给客户，并且不能出现横向滚动条。浏览器厂商会自作主张把布局视口

进行压缩（压缩的是css面积）

最终：我们的手机宽度此时用三种像素描述就变为了：

物理像素 750

设备独立像素 375

css像素 980

12、视觉视口

宽度永远是屏幕宽度所包含的css像素值

13、理想视口

meta name = "viewport" content = "width=device-width"

最终达到的目标 布局视口 = 视觉视口 = 理想视口 = 设备独立像素 = 375

14、meta笔记

meta(苹果发明的，桌面浏览器不支持)

15、pc缩放

布局

元素

16、移动端的用户缩放（草料使用）

草料二维码

17、放大有什么变化

布局 不变 布局视口大小不会发生改变

页面上的元素 面积变大 css像素值不变

元素占用物理像素 变多

视觉视口 变小

视觉视口中的元素个数 变少

18、缩小有什么变化

布局 不变 布局视口大小不会发生改变

元素 面积变小 css像素值不变

元素占用物理像素 变少

视觉视口 变大

视觉视口种的元素个数 变多

19、缩放笔记

20、meta解决大元素兼容性问题

meta标签同时设置 width = device-width 和 initial-scale=1.0

21、只设置initial-scale和只设置width效果一样

22、系统缩放有什么不同（会让布局视口根据视觉视口改变而改变）

会让布局视口根据视觉视口改变而改变

我们在考虑的时候，先根据用户的缩放去考虑，然后最后在基础上加上视觉视口的改变就ok；

23、同时initial-scale和width听谁的

如果同时设置，最终谁让我的布局视口更大，就听谁的；

24、最终我们以后用的meta标签要如何写

<meta name="viewport" content="width=device-width,initial-scale=1.0,user-scalable=no" />

25、meta元素设置的所有属性

###viewport

<meta name="viewport" content="" />

width [pixel\_value | device-width] width

-- 直接去设置具体数值大部分的安卓手机是不支持的 但是IOS支持

initial-scale 初始缩放比例

user-scalable 是否允许缩放 （no||yes）,默认允许

minimum-scale 允许缩放的最小比例

maximum-scale 允许缩放的最大比例

target-densitydpi

height

###width

就是用来控制布局视口的大小的，width=device-width会使布局视口的大小变成理想视口的大小（即独立设备像素代表的值）

95%的浏览器都支持width=device-width

###initial-scale

缩放是根据理想视口来计算的，这个缩放不同于我们用户的缩放，它会使布局视口跟随着我们的视觉视口一起转变

所以只设置inital-scale=1其实等同于只设置width=device-width

###完美视口！！！！

<meta name="viewport" content="width=device-width,initial-scale=1.0,user-scalable=no" /

如果你页面中存在太大的元素，你的meta标签只使用width=device-width，initial-scale=1.0中的一个，有些浏览器会扩展布局视口的宽度来容纳这个元素，这里的兼容性很复杂，但你两个都使用了，大部分浏览器都不会改变布局视口了

###width与inital-scale之间的冲突

布局视口在width与inital-scale产生分歧时会选择他们中比较大的那一个

###minimum-scale 允许缩放的最小比例

###maximum-scale 允许缩放的最大比例

没有这些指令，默认为20%-500%

有这些指令后可扩大到10%-1000%

安卓webkit不支持这两个属性（默认缩放范围为25%-400%）

ie压根不认识他们俩

# 2.适配

## 2.1 为什么要做适配

因为同样的一个元素，在一个设备上和另外一个设备上显示的布局不同。也就是比例不一样。导致不同设备显示不成正比

## 2.2 px em rem %单位的区别

px 绝对的一个单位，不能改变

em rem % 都是相对单位，都要有相对的基准值

em相对的是自身的字体大小

rem相对的永远是根元素html的字体大小 默认1rem = 16px

% 对于元素的宽和高相对的是父元素（只是爹）

## 2.3 rem适配（操作行内样式和操作文档内嵌样式）

### 2.3.1 rem适配方式：

我们可以动态的把屏幕的宽度设置为根元素字体大小;那么在不同的设备上，我们相当于拿到不同设备的宽度设置为html根元素的字体大小。也就是说在不同设置上都有这样的一个关系: 1rem = 设备的宽度.

//自调函数前面加!防止，浏览器解析认为函数和上面是一体的；

!(function(){

//拿到屏幕宽度

var width = document.documentElement.clientWidth;

//自己创建style标签

var styleNode = document.createElement('style');

//在style上添加样式

styleNode.innerHTML = 'html{font-size: '+ width/16 +'px !important;}';

//把style添加到head里面

document.head.appendChild(styleNode);

})();

### 2.3.2自调函数封装

自调函数：通常用在初始化，并且函数只执行一次的时候。习惯性在前面加！

### 2.3.3什么时候用rem适配

1、首先我们的设计图稿大于屏幕宽度 此时我们就要立马想到rem适配

2、在设计图稿上元素狠多

优点： 优点是可以使用meta完整的标签属性

缺点： 计算比较复杂; ===> less 考拉

## 2.4 viewport适配

### 2.4.1是什么，目的是什么

目的是将我们的屏幕视口的宽度调整到和图稿宽度一致

### 2.4.2什么时候要用到viewport适配

首先我们的设计图稿小于屏幕宽度,此时我们就要立马想到viewport适配，其实就是meta标签当中的initial-scale

优点：不用复杂的计算，直接使用图稿上标注的px值

缺点：不能使用完整的meta标签，并且常会出现图片失真；

### 2.4.3图稿: 300 屏幕: 375

计算initial-scale的值为多少，我们可以让屏幕宽度（布局视口）从375 变为 300

这个值比1大 然后才会让视口缩小 应该是375/300

!(function(){

var targetWidth = 300;

var width = document.documentElement.clientWidth;

var scale = width/targetWidth;

//获取上面的meta标签

var metaNode = document.querySelector('meta[name="viewport"]');

//给这个标签的属性更改成我们要设置的属性

metaNode.content = 'initial-scale='+ scale +',user-scalable=no'

})();

## 2.5 1物理像素适配

（其实就是rem适配和viewport适配的结合）

### 2.5.1目的

为了得到一个1物理像素的边框而出现的

### 2.5.2做法

盒子宽高使用rem 边框使用px 先根据dpr进行系统缩小相应倍数，再让rem进行反向放大相应倍数。

//将边框缩小的同时，其实把我们的盒子宽和高也缩小了相同的倍数，因此，我们需要反向乘回来

var width = document.documentElement.clientWidth;

var dpr = window.devicePixelRatio;

var scale = 1/dpr;

var metaNode = document.querySelector('meta[name="viewport"]');

metaNode.setAttribute('content','initial-scale='+ scale +',user-scalable=no');

//如果我们拿屏幕的宽度是在缩小之后拿的，其实拿到的宽度是之前的2倍，已经相当于乘以2了，下面设置的值就不需要再乘

//var width = document.documentElement.clientWidth;

var styleNode = document.createElement('style');

styleNode.innerHTML = 'html{font-size: '+ width/16\*dpr +'px !important;}'

document.head.appendChild(styleNode);

# 3. pc和移动端基础事件

//pc的事件

//dom0事件

box.onmousedown = function(event) {console.log('down');};

//dom2

box.addEventListener('mousedown', function() {});

box.onmousemove = function(event) {console.log('move');};

box.onmouseup = function(event) {console.log('up');};

//移动端

box.addEventListener('touchstart', function() {console.log('start')});

box.addEventListener('touchmove', function() {console.log('move')});

box.addEventListener('touchend', function() {console.log('end')});

# 4.移动端如何获取手指事件的位置

changedTouches:

目标元素目标事件上的手指列表,最终我们选择手指列表要的是changedTouches

targetTouches: 目标元素上的手指列表

touches: 屏幕上的手指列表

box.addEventListener('touchstart',function(event){

//直接在事件当中把手指先给拿到，以后用的时候，我们可以直接获取位置；

var touch = event.changedTouches[0];

console.log(event);

console.log(touch.clientX);

console.log(touch.clientY);

})

# 5.点透事件处理

## 5.1什么事点透(穿透)事件

在a标签上设置一个div遮罩层,给遮罩层添加点击事件,点击遮罩层隐藏.如果点击的是a标签的上方,遮罩层消失后会发生跳转.

## 5.2解决方法

//第一步，先把浏览器默认行为禁止掉，a标签的跳转行为就被剁了

document.addEventListener('touchstart',function(event){

event.preventDefault();

})

//第二步，拿到页面上所有的a标签，一个一个添加点击事件

var aNodes = document.querySelectorAll('a');

for(var i = 0; i < aNodes.length; i++){

aNodes[i].addEventListener('touchstart',function(){

//更改浏览器地址

window.location.href = this.href;

})

}

# 6.移动端准备7项工作

1. meta

<meta name="viewport" content="width=device-width,initial-scale=1,user-scalable=no" />

2. css初始化

\* {

margin: 0;

padding: 0;

}

img {

vertical-align: middle;

}

ul {

list-style: none;

}

a {

text-decoration: none;

-webkit-tap-highlight-color: rgba(0, 0, 0, 0);

}

input {

outline: none;

}

3.适配

!(function() {

var width = document.documentElement.clientWidth;

var styleNode = document.createElement('style');

styleNode.innerHTML = 'html{font-size: ' + width / 16 + 'px;}'

document.head.appendChild(styleNode);

})();

!(function() {

var targetWidth = 300;

var width = document.documentElement.clientWidth;

var scale = width / targetWidth;

var meta = document.querySelector('meta[name="viewport"]');

meta.setAttribute('content', 'initial-scale=' + scale + ',user-scalable=no');

})();

4.阻止浏览器默认行为

document.addEventListener('touchstart', function(event) {

event.preventDefault();

});

5.处理点透事件

var aNodes = document.querySelectorAll('a');

for(var i = 0; i < aNodes.length; i++) {

aNodes[i].addEventListener('touchstart', function() {

window.location.href = this.href;

})

}

6.禁止系统滚动条

html,body {

height: 100%;

overflow: hidden;

}

7.模拟屏幕

#wrap {

width: 100%;

height: 100%;

overflow: hidden;

}