# 移动端事件

## 1. PC端事件在移动端的兼容问题

### 1.1 click事件的200~300ms延迟问题

由于移动端默认的布局视口宽度是980像素，所以网页文字非常小，为了快速让网页还原到原来的大小，Safari最新引入了双击缩放功能：用户双击手机页面的时候，浏览器会智能的缩放当前页面到原始大小。

​双击缩放的原理就是，当用户click一次之后，浏览器会经过约300ms之后检测是否再有一次click，如果有的话，就会缩放页面。否则的话就是一个click事件。

由于双击缩放功能存在，click事件触发就会有大约200~300ms的延迟。

### 1.2 dblclick事件失效

由于双击缩放的存在，pc端的dblclick事件也失效了。

## 2. 移动端特有的touch事件

由于移动端设备大都具备触摸功能，所以移动端浏览器都引入了触摸(touch)事件。

touch相关的事件跟普通的其他dom事件一样使用，可以直接用addEventListener来监听和处理。

最基本的touch事件包括4个事件：

1. touchstart： 当在屏幕上按下手指时触发
2. touchmove： 当在屏幕上移动手指时触发
3. touchend： 当在屏幕上抬起手指时触发
4. touchcancel 当一些更高级别的事件发生的时候（如电话接入或者弹出信息）会取消当前的touch操作，即触发touchcancel。一般会在touchcancel时暂停游戏、存档等操作。

### 2.1 touch事件与click事件同时触发

在很多情况下，触摸事件和鼠标事件会同时被触发（目的是让没有对触摸设备优化的代码仍然可以在触摸设备上正常工作）。

因为双击缩放检测的存在，在移动设备屏幕上点击操作的事件执行顺序：

**touchstart(瞬间触发) → touchend → click(200-300ms延迟)**

如果你使用了触摸事件，可以调用 event.preventDefault()来阻止鼠标事件被触发。

### 2.2 touchstart事件

​ 当用户手指触摸到的触摸屏的时候触发。事件对象的 target 就是touch 发生位置的那个元素。

<div>

点击我！

</div>

<script>

var box = document.querySelector("div");

box.addEventListener("touchstart", function (e) {

console.log('touchstart');

});

</script>

### 2.3 touchmove事件

当用户在触摸屏上移动触点(手指)的时候，触发这个事件。一定是先要触发touchstart事件，再有可能触发 touchmove 事件。

​touchmove 事件的target 与最先触发的 touchstart 的 target 保持一致。touchmove事件和鼠标的mousemove事件一样都会多次重复调用，所以，事件处理时不能有太多耗时操作。不同的设备，移动同样的距离 touchmove 事件的触发频率是不同的。

注意：

1. 即使手指移出了 原来的target 元素，则 touchmove 仍然会被一直触发，而且 target 仍然是原来的 target 元素。
2. touchmove事件会多次重复触发，由于移动端计算资源宝贵，尽量保证[事件节流](https://link.jianshu.com?t=https%3A%2F%2Fwww.2cto.com%2Fkf%2F201411%2F352213.html)

<div>

<p></p>

</div>

<script>

var i = 1;

var box = document.querySelector("div");

var p = document.querySelector("p");

box.addEventListener("touchmove", function (e){

p.innerHTML = e.target.tagName + ", " + i++;

})

</script>

### 2.4 touchend事件

​ 当用户的手指抬起的时候，会触发 touchend 事件。如何用户的手指从触屏设备的边缘移出了触屏设备，也会触发 touchend 事件。

touchend 事件的 target 也是与 touchstart 的 target 一致，即使已经移出了元素。

一次完整的touch事件的触发顺序和过程

### 2.5 touchcancel事件

​ 当触点由于某些原因被中断时触发。有几种可能的原因如下(具体的原因根据不同的设备和浏览器有所不同):

* 由于某个事件取消了触摸：例如触摸过程被一个模态的弹出框打断。
* 触点离开了文档窗口，而进入了浏览器的界面元素、插件或者其他外部内容区域。
* 当用户产生的触点个数超过了设备支持的个数，从而导致 TouchList 中最早的 Touch对象被取消

touchcancel 事件一般用于保存现场数据。比如：正在玩游戏，如果发生了 。touchcancel 事件，则应该把游戏当前状态相关的一些数据保存起来。

## 3. 触摸事件对象

TouchEvent 是一类描述手指在触摸平面（触摸屏、触摸板等）的状态变化的事件。这类事件用于描述一个或多个触点，使开发者可以检测触点的移动，触点的增加和减少，等等。

每 个 [Touch](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouch) 对象代表一个触点; 每个触点都由其位置，大小，形状，压力大小，和目标 [element](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FElement) 描述。 [TouchList](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchList) 对象代表多个触点的一个列表.

### 3.1 TouchEvent

[*TouchEvent*](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchEvent)的属性继承了 [*UIEvent*](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FUIEvent) 和 [*Event*](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FEvent)。

属性列表：

1. [TouchEvent.changedTouches](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchEvent%2FchangedTouches)： 一个 [TouchList](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchList) 对象，包含了代表所有从上一次触摸事件到此次事件过程中，状态发生了改变的触点的 [Touch](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouch) 对象。
2. [TouchEvent.targetTouches](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchEvent%2FtargetTouches)： 一个 [TouchList](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchList) 对象，是包含了如下触点的 [Touch](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouch) 对象：触摸起始于当前事件的目标 [element](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FElement) 上，并且仍然没有离开触摸平面的触点。
3. [TouchEvent.touches](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchEvent%2Ftouches)： 一 个 [TouchList](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouchList) 对象，包含了所有当前接触触摸平面的触点的 [Touch](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FTouch) 对象，无论它们的起始于哪个 [element](https://link.jianshu.com?t=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FAPI%2FElement) 上，也无论它们状态是否发生了变化。

<style>

.box {

width: 100px;

height: 100px;

border: 1px solid #09c;

background-color: #0dc;

}

</style>

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

<script>

window.onload = function() {

var box = document.querySelector('.box');

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

console.dir(e); // 查看TouchEvent对象的属性和方法

});

}

</script>

### 3.2 TouchList详解

​ 一个TouchList代表一个触摸屏幕上所有触点的列表。

​ 举例来讲, 如果一个用户用三根手指接触屏幕(或者触控板), 与之相关的TouchList 对于每根手指都会生成一个 Touch对象, 共计 3 个.

1. 只读属性：length

返回这个TouchList中Touch对的个数。(就是有几个手指接触到了屏幕)

1. 方法：item(index)

返回TouchList中指定索引的Touch对象。

<div>

<p style="font-size: 50px; color: #ffffff;"></p>

</div>

<script>

var box = document.querySelector("div");

var p = document.querySelector("p");

box.addEventListener("touchend", function (e){

p.innerHTML = e.changedTouches.length; //返回Touch对象的个数

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

//遍历出来每个Touch对象

console.log(e.changedTouches.item(i));

}

})

</script>

**测试多个手机触摸屏幕：**

<div></div>

<p></p>

<script>

var div = document.querySelector("div");

var p = document.querySelector("p");

div.addEventListener("touchstart", function (e){

var msg = "touches.length: " + e.touches.length +

"<br> targetTouches.length: " + e.targetTouches.length +

"<br> changedTouches.length: " + e.changedTouches.length;

p.innerHTML = msg;

})

</script>

操作：

1. 放1个手指在div上
2. 先放1个手指在其他地方，然后再放1个手指在div上
3. 先放1个手指在其他地方，然后再逐渐放2个手指在div上

## 3.3 Touch详解

​ Touch表示用户和触摸设备之间接触时单独的交互点(a single point of contact)。​ 这个交互点通常是一个手指或者触摸笔，​ 触摸设备通常是触摸屏或者触摸板。

基本属性列表（都是只读）:

| **编号** | **属性名** | **属性说明** |
| --- | --- | --- |
| 1. | identifier | 表示每 1 个 Touch 对象 的独一无二的 identifier。有了这个 identifier 可以确保你总能追踪到这个 Touch对象。 |
| 2. | screenX | 触摸点相对于屏幕左边缘的 x 坐标。 |
| 3. | screenY | 触摸点相对于屏幕上边缘的 y 坐标。 |
| 4. | clientX | 触摸点相对于浏览器的 viewport左边缘的 x 坐标。不会包括左边的滚动距离。 |
| 5. | clientY | 触摸点相对于浏览器的 viewport上边缘的 y 坐标。不会包括上边的滚动距离。 |
| 6. | pageX | 触摸点相对于 document的左边缘的 x 坐标。 与 clientX 不同的是，他包括左边滚动的距离，如果有的话。 |
| 7. | pageY | 触摸点相对于 document的左边缘的 y 坐标。 与 clientY 不同的是，他包括上边滚动的距离，如果有的话。 |
| 8. | target | 总是表示 手指最开始放在触摸设备上的触发点所在位置的 element。 即使已经移出了元素甚至移出了document, 他表示的element仍然不变 |

案例：

var box = document.querySelector("div");

var p = document.querySelector("p");

box.ontouchstart = function (e){

var touchList = e.changedTouches;

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

var touch = touchList[i];

var msg = `id : ${touch.identifier} <br>

screenX : ${touch.screenX} <br>

screenY : ${touch.screenY} <br>

clientX : ${touch.clientX} <br>

clientY : ${touch.clientY} <br>

pageX : ${touch.pageX} <br>

pageY : ${touch.pageY} <br>

target: ${touch.target.nodeName} <br>

`;

p.innerHTML = msg;

}

}

没有左右滚动：

左右滚动：pageX 明显大于 clientX

## 4. 封装移动端tap事件

由于点击事件经常使用，如果用click会有延迟问题，一般我们会用touch事件模拟移动端的点击事件, 以下是封装的几个事件，仅供参考。

(function (window){ //传入window，提高变量的查找效率

function myQuery(selector){ //这个函数就是对外提供的接口。

//调用这个函数的原型对象上的\_init方法，并返回

return myQuery.prototype.\_init(selector);

}

myQuery.prototype = {

/\*初始化方法，获取当前query对象的方法\*/

\_init: function (selector){

if (typeof selector == "string"){

//把查找到的元素存入到这个原型对象上。

this.ele = window.document.querySelector(selector);

//返回值其实就是原型对象。

return this;

}

},

/\*单击事件：

\* 为了规避click的300ms的延迟，自定义一个单击事件

\* 触发时间：

\* 当抬起手指的时候触发

\* 需要判断手指落下和手指抬起的事件间隔，如果小于500ms表示单击时间。

\* 如果是大于等于500ms，算是长按时间

\* \*/

tap: function (handler){

this.ele.addEventListener("touchstart", touchFn);

this.ele.addEventListener("touchend", touchFn);

var startTime,

endTime;

function touchFn(e){

e.preventDefault()

switch (e.type){

case "touchstart":

startTime = new Date().getTime();

break;

case "touchend":

endTime = new Date().getTime();

if (endTime - startTime < 500){

handler.call(this, e);

}

break;

}

}

},

/\*\*

\* 长按

\* @param handler

\*/

longTag: function (handler){

this.ele.addEventListener("touchstart", touchFn);

this.ele.addEventListener("touchmove", touchFn);

this.ele.addEventListener("touchend", touchFn);

var timerId;

function touchFn(e){

switch (e.type){

case "touchstart" : //500ms之后执行

timerId = setTimeout(function (){

handler.call(this, e);

}, 500)

break;

case "touchmove" :

//如果中间有移动也清除定时器

clearTimeout(timerId)

break;

case "touchend" :

//如果在500ms之内抬起了手指，则需要定时器

clearTimeout(timerId);

break;

}

}

},

/\*\*

\* 左侧滑动。

\* 记录手指按下的左边，在离开的时候计算 deltaX是否满足左滑的条件

\*/

slideLeft: function (handler){

this.ele.addEventListener("touchstart", touchFn);

this.ele.addEventListener("touchend", touchFn);

var startX, startY, endX, endY;

function touchFn(e){

e.preventDefault();

var firstTouch = e.changedTouches[0];

switch (e.type){

case "touchstart":

startX = firstTouch.pageX;

startY = firstTouch.pageY;

break;

case "touchend":

endX = firstTouch.pageX;

endY = firstTouch.pageY;

//x方向移动大于y方向的移动，并且x方向的移动大于25个像素，表示在向左侧滑动

if (Math.abs(endX - startX) >= Math.abs(endY - startY) && startX - endX >= 25){

handler.call(this, e);

}

break;

}

}

},

/\* 右侧滑动 \*/

rightLeft: function (e){

//TODO:

}

}

window.$ = window.myQuery = myQuery;

})(window);

// ========================

// 使用：

$("div").tap(function (e){

console.log("单击事件")

})

$("div").longTag(function (){

console.log("长按事件");

})

$("div").slideLeft(function (e){

console.log(this);

this.innerHTML = "左侧滑动了....."

})

## 5. 触摸手势封装相关的框架及事件

手势相关的事件一般就是tap类（触屏）和滑动（swipe)事件两类。都是基于原生的touchstart、touchmove、touchend事件，封装成不同的手势类型自定义事件。

### 5.1 tap类事件

触碰事件，我目前还不知道它和touch的区别，一般用于代替click事件，有tap longTap singleTap doubleTap四种之分。

1. tap: 手指碰一下屏幕会触发
2. longTap: 手指长按屏幕会触发
3. singleTap: 手指碰一下屏幕会触发
4. doubleTap: 手指双击屏幕会触发

### 5.2 swipe类事件

滑动事件，有swipe swipeLeft swipeRight swipeUp swipeDown 五种之分。

1. swipe：手指在屏幕上滑动时会触发
2. swipeLeft：手指在屏幕上向左滑动时会触发
3. swipeRight：手指在屏幕上向右滑动时会触发
4. swipeUp：手指在屏幕上向上滑动时会触发
5. swipeDown：手指在屏幕上向下滑动时会触发

### 5.3 zepto的手势相关事件

[Zepto.js](https://link.jianshu.com?t=http%3A%2F%2Fwww.zeptojs.cn%2F) 是一个轻量级的针对现代高级浏览器的JavaScript库， 它适配了jQuery的大部分api，也就是jQuery怎么用，Zepto.js就怎么用。它非常小，非常适合移动端。

Zepto.js的touch模块中封装了手势相关的代码。封装了再触摸设备上触发tap– 和 swipe– 相关事件，也适用于所有的touch(iOS, Android)和pointer事件(Windows Phone)。

* 触屏事件：tap、singleTap、doubleTap、longTap(>750ms）
* 滑动事件：swipe、swipeLeft,、swipeRight,、swipeUp,、swipeDown

<style>.delete { display: none; }</style>

<ul id=items>

<li>List item 1 <span class=delete>DELETE</span></li>

<li>List item 2 <span class=delete>DELETE</span></li>

</ul>

<script>

$('#items li').swipe(function(){

$('.delete').hide()

$('.delete', this).show()

})

$('.delete').tap(function(){

$(this).parent('li').remove()

})

</script>

### 5.4 其他移动端手势相关库

1. [百度云的touch.js](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2FClouda-team%2Ftouchjs)
2. [hammer.js](https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fhammerjs%2Fhammer.js)  
   hammer提供了不仅仅tap、swipe等事件，还提供了：pan（平移)、pinch类（捏拿缩放）、 press类（按住）、 rotate类（旋转）类手势支持， [hammer.js详解教程](https://link.jianshu.com?t=http%3A%2F%2Fwww.cnblogs.com%2Fimwtr%2Fp%2F5884484.html)

## 6. 移动端点击穿透问题

如果某个返回按钮的位置，恰好在要返回的这个页面的带有href属性的a标签的范围内，在点击返回按钮后，页面快速切换到有a标签的页面，300ms后触发了click事件，从而触发了a标签的意外跳转，这个就是典型的点击穿透问题。罪魁祸首其实就是a标签跳转默认是click事件触发，而移动端的touch事件触发之后，依然会在300ms后触发click事件。

解决办法：  
1.就是阻止触发touch事件完成后的click事件。  
2.不要混用touch和click事件。显然不可能都绑定click事件，因为要解决300ms延迟问题(除了fastclick)，那么只能都绑定touch事件，这样click事件永远不会被触发。

注意：zepto并没有阻止click事件，所以使用zepto的tap事件依然会导致点击穿透问题，你需要手动添加 e.preventDefault() 来阻止click事件

作者：IT老马  
链接：https://www.jianshu.com/p/997b23232bb8  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# 移动端的touch click事件的理解+点透

移动端在touch上一共有4个事件

touchstart touchmove touchend touchcancel， touchcancel, 一般来说，它们执行的顺序为 touchstart -> touchmove -> touchend -> touchcancel . 其中touchcancel一般情况下不会触发，也不是这里讨论的焦点。

这里会结合click对上面的事件进行讨论, touch发生在click之前

先上段代码，直观感受一下

<!DOCTYPE html>

<html>

<head>

<style type="text/css">

#level0 {

/\* width: 500px;

height: 500px; \*/

}

#level1-0 {

background: red;

width: 500px;

height: 500px;

}

#level1-1 {

background: green;

width: 500px;

height: 500px;

}

</style>

</head>

<body>

<div id="level0">

<div id="level1-0">

</div>

<div id="level1-1">

</div>

</div>

</body>

<script type="text/javascript">

var level10 = document.getElementById("level1-0");

level10.addEventListener('touchstart', function(e) {

console.log(1);

});

level10.addEventListener('touchmove', function(e) {

console.log(2);

});

level10.addEventListener('touchend', function(e) {

console.log(3);

});

level10.onclick = function() {

console.log(5);

}

document.body.onclick = function() {

console.log('6');

}

</script>

</html>

在红色区域点击会出现什么效果呢？ 出现的是 1 3 5 6， 奇怪了 touchmove 为何不执行，因为我们并没有移动，也就是说，必须触碰到屏幕上面，而且发生了移动动作，touchmove才执行，现在我们触碰到，而且手指稍微动一下，发现输出的效果是， 1 2(+) 3, 其中touchmove 可能触发多次，又奇怪了， click为何不执行， 因为 click执行的条件是 **点击， 而且不移动** 所以一般情况下，我们可以理解成 touchmove和click是相斥的。

我们知道，当一个用户在点击屏幕的时候，系统会触发touch事件和click事件，touch事件优先处理，touch事件经过 捕获，处理, 冒泡 一系列流程处理完成后， 才回去触发click事件

既然touch事件和click事件有了优先级别，那么能不能在touch阶段取消掉系统触发的click事件呢？当然是可以的，浏览器提供了这样的能力。**在touch事件里面，调用e.preventDefault() 就可以阻止本次点击系统触发的click事件，即本次相关的click都不会执行**

把上面代码稍微加一点

level10.addEventListener('touchstart', function(e) {

console.log(1);

e.preventDefault();

});

点击的时候 发现 只有 1 3， 说明click被阻止了，当然在touchend里面加效果也一样，所以 在touch事件里面加 e.preventDefault可以取消系统产生的click事件， 当然不会阻止后面的touch事件。

**用个具体的例子看看 如何解决点透问题**

产生点透问题的原因， 可以先看看代码吧

<!DOCTYPE html>

<html>

<head>

<style type="text/css">

#level0 {

/\* width: 500px;

height: 500px; \*/

position: relative;

}

#level1-0 {

position: absolute;

z-index: 1;

background: red;

width: 500px;

height: 500px;

}

#level1-1 {

background: green;

width: 500px;

height: 500px;

}

</style>

</head>

<body>

<div id="level0">

<div id="level1-0">

</div>

<div id="level1-1">

</div>

</div>

</body>

<script type="text/javascript">

var level10 = document.getElementById("level1-0");

var level11 = document.getElementById("level1-1");

level10.addEventListener('touchstart', function(e) {

level10.style.display = 'none';

});

level11.onclick = function() {

console.log('level11莫名被点击了');

}

</script>

</html>

**本来是 level1-0 和 level1-1是兄弟节点，即他们之间不会发生什么 事件传递**， 目前level1-0相当于一个覆盖层，覆盖在level1-1上面， 按理说点击 level1-0的时候，level1-0会阻挡所有的事件，事件不会传递给level1-1，当点击level1-0的时候，实际上level1-1也发生了点击事件，即上面的输出结果为  
level1-0消失， 输出 level11莫名被点击了, 这就是点透

点透发生的条件，

1. A 和 B不是后代继承关系(如果是后代继承关系的话，就直接是冒泡子类的话题了)
2. A发生touch， A touch后立即消失， B事件绑定click
3. A z-index大于B，即A显示在B浮层之上

**点透发生的理由**: 当手指触摸到屏幕的时候，系统生成两个事件，一个是touch 一个是click，touch先执行，touch执行完成后，A从文档树上面消失了，而且由于移动端click还有延迟200-300ms的关系，当系统要触发click的时候，发现在用户点击的位置上面，目前离用户最近的元素是B，所以就直接把click事件作用在B元素上面了.

那如何才能解决点透问题呢？ 还记得我之前说过么，系统提供了先触发的touch事件去取消系统生成的click事件，所以只要在touch事件的某个处理函数中 执行 e.preverDefault即可， 一般我们在touchend中执行

在上面代码中，加上这句就完美解决了

level10.addEventListener('touchend', function(e) {

e.preventDefault();

});

当然点透问题，还有其他的解决方法，关键是 要么是需求本次系统生成的click事件，要么是当系统触发click的时候，当前的触发touch的那个dom节点还存在。比如将其一延迟3s在关闭

setTimeout(() => {

level10.style.display = 'none';

}, 300);

作者：yongningfu  
链接：https://www.jianshu.com/p/dc3bceb10dbb  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

# 适配

**像素**

在移动端给一个元素设置 width:200px 时发生了什么？这里的px到底是多长呢？像素是网页布局的基础，但是我们一直在用直觉使用它。

其实存在两种像素：

**1. 设备像素**

屏幕的物理像素，任何设备屏幕的物理像素的数量都是固定不变的，单位是pt。

**2. CSS像素**

在CSS、JS中使用的一个抽象的概念，单位是 px。

顺便说下，CSS像素也可以称为**设备独立像素**(device-independent pixels)，简称为dips，单位是dp。

那么，我们现在再来说说一个元素 width:200px 以后会怎么样。这个元素跨越了200个CSS元素，200个CSS元素相当于多少个设备像素取决于两个条件：

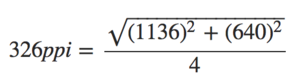
* 页面是否缩放
* 屏幕是否为高密度

这两方面后面再解释，先梳理一下手机硬件之间的关系，注意这里使用的都是**物理像素**。

以 iPhone5 为例，我们已知的是：

1. **分辨率**：1136pt x 640pt  
   指屏幕上垂直有 1136 个物理像素，水平有 640 个物理像素
2. **屏幕尺寸**： 4英寸  
   注意英寸是长度单位，不是面积单位。4英寸指的是屏幕对角线的长度。
3. **屏幕像素密度**：326dpi  
   屏幕像素密度（Pibel Per Inch）简称 ppi ，单位是 dpi（dot per inch）。这里指屏幕水平或垂直每英寸有326个物理像素。原则上来说，ppi越高越好，因为图像会更加细腻清晰。

ppi 是可以通过 **分辨率** 和 **屏幕尺寸** 计算得到的：

[](https://camo.githubusercontent.com/c1502f74d9951713cd06f2fd56ba2937a374d92a/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f31362d312d31392f35363932343537332e6a70673f696d616765566965772f322f772f333030)

[这个网站](https://www.sven.de/dpi/)列出了很多设备的**分辨率**和**屏幕尺寸**，并且计算了**ppi**。

**视口**

桌面浏览器中，浏览器窗口就是约束你的CSS布局视口（又称初始包含块）。它是所有CSS百分比宽度推算的根源，它的作用是给**CSS布局限制了一个最大宽度**，视口的宽度和浏览器窗口宽度一致。

但是在移动端，情况就很复杂了。

**布局视口**

一个没有为移动端做优化的网页，会尽可能缩小网页让用户看到所有东西。这是我的手机查看[博客园](http://www.cnblogs.com/)的样子，你也可以在Chrome中以移动开发模式看到。

[](https://camo.githubusercontent.com/204ead9fb51cda7871a1847de0f3ea5cc6e266a9/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466c762d6c39696366612d307853796e69665f6659707472786c58373f696d616765566965772f322f772f333030)

浏览器厂商为了让用户在小屏幕下网页也能够显示地很好，所以把视口宽度设置地很大，一般在 768px ~ 1024px 之间，最常见的宽度是 980px。

所以，在手机上，视口与移动端浏览器屏幕宽度不再相关联，是完全独立的，这个浏览器厂商定的视口被称为**布局视口**。

[](https://camo.githubusercontent.com/43e1321a2708326ba26ac9710f714f75838b5343/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f46764754754e6a6a366a6270485a765874636a2d4c365f7951577858)

布局视口我们是看不见的，只知道网页的最大宽度是 980px ，并且被缩放在了屏幕内。

可以这样设置布局视口的宽度：

<meta name="viewport" content="width=640">

**媒体查询与布局视口**

700px 指的是布局视口的宽度

@media (min-width: 700px){

...

}

document.documentElement.clientWidth/Height返回布局视口的尺寸

**视觉视口**

**视觉视口**是用户正在看到的网页的区域，大小是屏幕中CSS像素的数量。

[](https://camo.githubusercontent.com/7caa987d3754bd966fab21333dc8a351a3d108b2/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466d786c6e414f5f5f2d35365a5a4e5335485a6364516c6f4e546649)

window.innerWidth/Height返回视觉视口的尺寸

**理想视口**

布局视口明显对用户是不友好的，完全忽略了手机本身的尺寸。所以苹果引入了理想视口的概念，它是**对设备来说最理想的布局视口尺寸**。理想视口中的网页用户最理想的宽度，用户进入页面的时候不需要缩放。

现在讨论所谓的『最理想的宽度』到底是多少？其实，如果我们把布局视口的宽度改成屏幕的宽度不就不用缩放了么。可以这样设置告诉浏览器使用它的理想视口：

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

定义理想视口是浏览器的事情，并不能简单地认为是开发者定义的，开发者只能使用。

screen.width/height返回理想视口的尺寸，有严重的兼容性问题---可能返回两种值：

1. 理想视口的尺寸（下载浏览器）
2. 屏幕的设备像素尺寸（内置浏览器）

[Screen size tests](http://quirksmode.org/m/tests/widthtest.html)和[Understanding viewport](http://andreasbovens.github.io/understanding-viewport/)可以测试你的设备的screen.width值，同一设备的不同浏览器返回的值可能是不一样的。这一情况主要发生在默认浏览器和下载浏览器（如UC、Chrome）之间。

默认浏览器是安卓系统内置的浏览器，长下面那个样子。而且它使用的是Webkit而不是Blink。只有在更新安卓系统的时候才能更新它。直到安卓4.3，Google不再更新。

[](https://camo.githubusercontent.com/d82c81999a80394c8a7d3126f756397066f7b9b5/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466e68394c6a7644566e4147556b745542496e633034514949595f49)

而下载浏览器都返回的是理想视口尺寸。

**缩放**

**缩放与设备像素、CSS像素的关系**

缩放是在放大或缩小CSS像素，比如一个宽度为 200px 的元素无论放大，还是200个CSS像素。但是因为这些像素被放大了，所以CSS像素也就跨越了更多的设备像素。缩小则相反。

**缩放与视口**

**缩放会影响视觉视口的尺寸**

页面被用户放大，视觉视口内CSS像素数量减少；被用户缩小，视觉视口内CSS像素数量增多就行了。这个道理应该是不难想的。

**用户缩放不会影响布局视口**

注意，这是『用户缩放』，后面会说开发者设置缩放的情况

**缩放比例**

我们在开发者工具中可以在这里查看缩放比例：

[](https://camo.githubusercontent.com/76598ea4df1f7c1476cf224c6ec72669cf766a47/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f31362d312d31392f39333230373435342e6a7067)

这里的 0.3 是相对于**理想视口**的。

在下载浏览器中，可以这么算（理想视口与视觉视口的比）：

zoom level = screen.width / window.innerWidth

**禁止缩放**

<meta name="viewport" content="user-scalable=no">

**设置缩放**

<meta name="viewport" content="initial-scale=2">

使用initial-scale有一个副作用：同时也会将布局视口的尺寸设置为缩放后的尺寸。所以initial-scale=1与width=device-width的效果是一样的。

**完美视口**

解决各种浏览器兼容问题的理想视口设置

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

**设备像素比**

在谈到像素的时候，讲到除了缩放，**屏幕是否为高密度**也会影响设备像素和CSS像素的关系。

在**缩放程度为100%**（这个条件很重要，因为缩放也会影响他们）时，他们的比例叫做**设备像素比**(device pixel ratio)：

dpr = 设备像素 / CSS像素

可以通过JS得到： window.devicePixelRatio

设备像素比也和视口有关：

dpr = 屏幕横向设备像素 / 理想视口的宽

**总结**

这一篇介绍了移动端适配需要掌握的知识，先说明了移动端存在的两种像素，然后介绍了三种视口，由缩放对视口的影响引入理想视口，最后说明设备像素比。下一篇介绍现在市面上的适配方案。

之前做过PC页面的人聊的最多的就是『兼容』，这是因为浏览器之间的差异引起的，不再多说。而移动端是基本没有『兼容』的问题的，全是CSS3，简直不要太开心。可是『适配』问题随之而来。

什么是『适配』？做PC页面的时候，我们按照设计图的尺寸来就好，这个侧边栏200px，那个按钮50px的。可是，当我们开始做移动端页面的时候，设计师给了一份宽度为640px的设计图。那么，我们把这份设计图实现在各个手机上的过程就是『适配』。

那么，我们怎么开始呢？目前有三种方法：

* 固定高度，宽度自适应
* 固定宽度，viewport缩放
* rem做宽度，viewport缩放

这三种方法的核心都是**视口的确定**，现在以实现这个设计图为例说明。

[](https://camo.githubusercontent.com/44836cf0098a50f84db0b6db69dc768cdd629c1a/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466c49715569627768566342346476522d3852504f546b6f67336b50)

**固定高度，宽度自适应**

[demo](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-fixed-height.html)

这也是目前使用最多的方法，垂直方向用定值，水平方向用百分比、定值、flex都行。[腾讯](http://xw.qq.com/index.htm)、[京东](http://m.jd.com/)、[百度](https://www.baidu.com/)、[天猫](https://www.tmall.com/)、[亚马逊](http://www.amazon.cn/)的首页都是使用的这种方法。

随着屏幕宽度变化，页面也会跟着变化，效果就和PC页面的流体布局差不多，在哪个宽度需要调整的时候使用\_响应式布局\_调调就行（比如[网易新闻](http://news.163.com/mobile/)），这样就实现了『适配』。

**原理**

这种方法使用了完美视口：

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

这样设置之后，我们就可以不用管手机屏幕的尺寸进行开发了。

**固定宽度，viewport缩放**

[demo](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-fixed-width.html)

设计图、页面宽度、viewport width使用一个宽度，浏览器帮我们完成缩放。单位使用px即可。

目前已知[荔枝FM](http://m.lizhi.fm/)、[网易新闻](http://c.3g.163.com/CreditMarket/default.html)在使用这种方法。有兴趣的同学可以看看是怎么做的。

**原理**

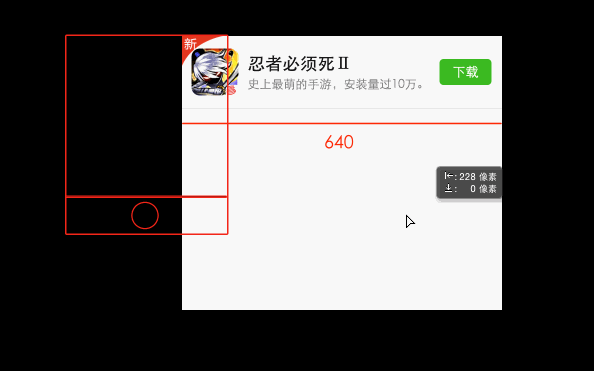
这种方法需要根据屏幕宽度来动态生成viewport，生成的 viewport 基本是这样：

<meta name="viewport" content="width=640,initial-scale=0.5,maximum-scale=0.5,minimum-scale=0.5,user-scalable=no">

640 是我们根据设计图定下的，0.5 是根据屏幕宽度动态生成的。

生成的viewport告诉浏览器网页的布局视口使用 640px，然后把页面缩放成50%，这是绝对的等比例缩放。图片、文字等等所有元素都被缩放在手机屏幕中。

这个gif图说明了一切：

[](https://camo.githubusercontent.com/3ff4622fe648d152ded003965e213a9258cf1981/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466a44437068786e3939306b443947544d4a5a4f59584a3566564b68)

**rem做宽度，viewport缩放**

[demo](http://www.meow.re/demo/screen-adaptation-in-mobileweb/app-rem.html)

这也是[淘宝](https://m.taobao.com)使用的方案，根据屏幕宽度设定 rem 值，需要适配的元素都使用 rem 为单位，不需要适配的元素还是使用 px 为单位。

具体使用方法见[使用Flexible实现手淘H5页面的终端适配](https://github.com/amfe/article/issues/17)

上文提供了sass和postcss的px2rem转换方法，我这里给出less的px2rem。因为less不支持函数，所以需要安装插件 [less-plugin-functions](https://github.com/seven-phases-max/less-plugin-functions) ，然后就简单了：

.function{

.px2rem(@px,@base:72px){

return: @px / @base \* 1rem;

}

}

这样使用:

div{

width: px2rem(100px);

}

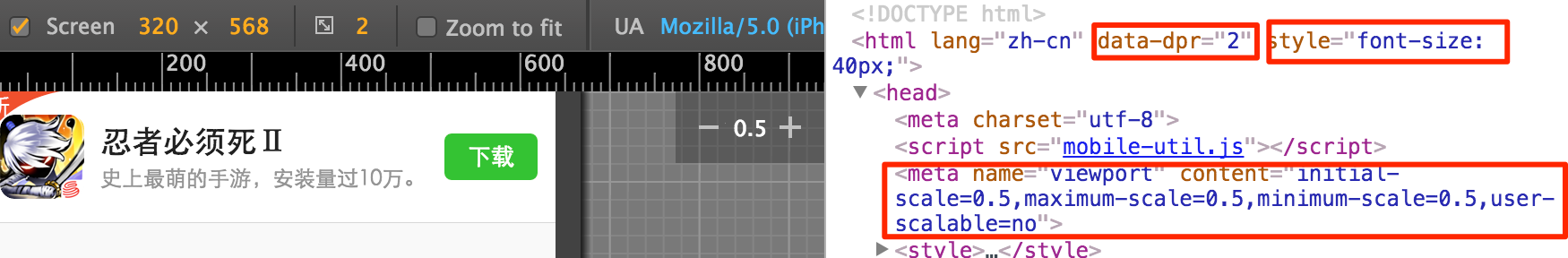
使用这个方法的库：

* [lib-flexible](https://github.com/amfe/lib-flexible)
* [hotcss](https://github.com/imochen/hotcss)

**原理**

实际上做了这几件事情：

1. 动态生成 viewport
2. 屏幕宽度设置 rem的大小，即给<html>设置font-size
3. 根据设备像素比（window.devicePixelRatio）给<html>设置data-dpr

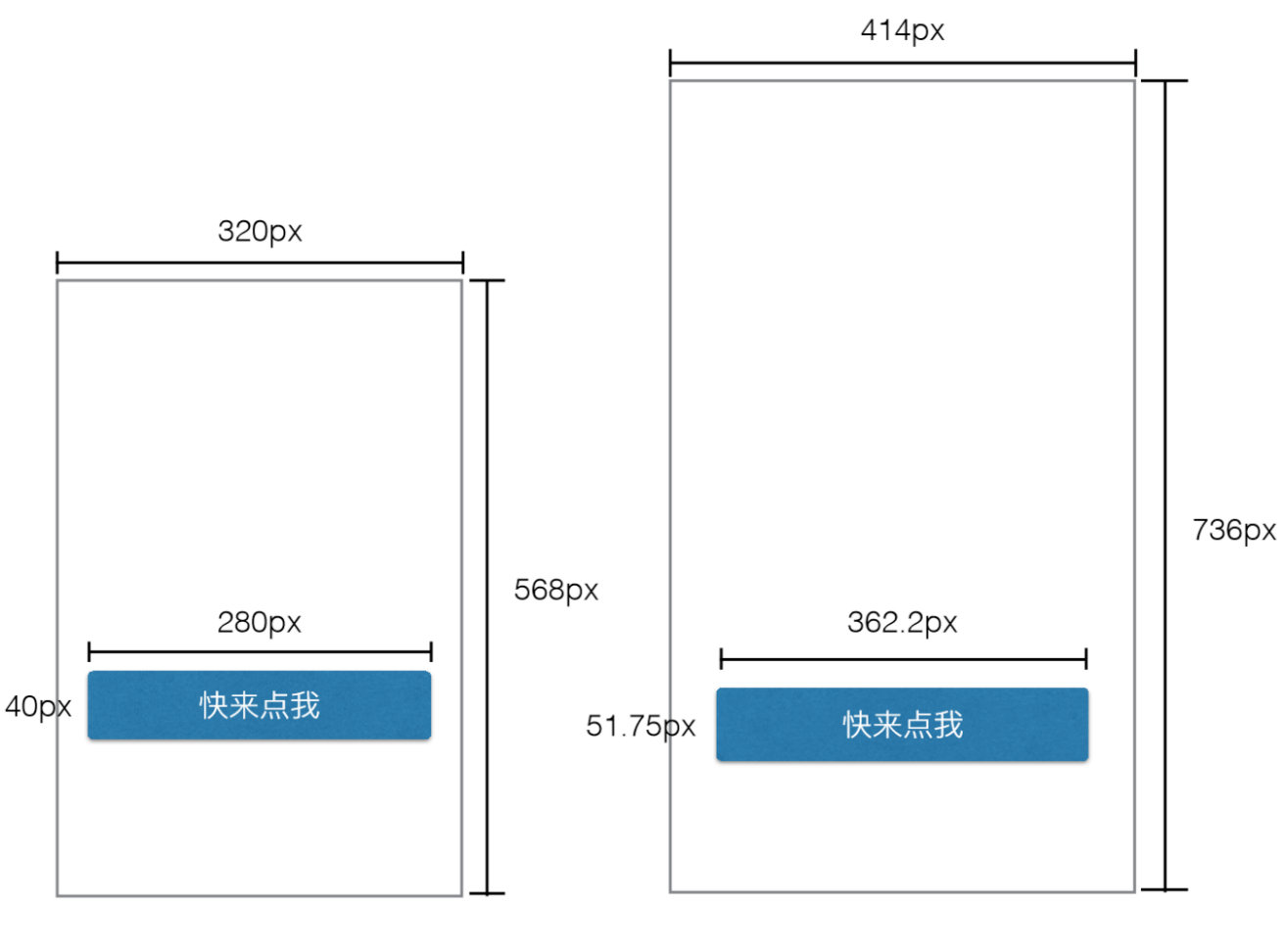
[](https://camo.githubusercontent.com/8b2ca71b169202a8e356d431c81a7b15ff6fdf42/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f4669455233715232717272584c6e61334b4e757a6d2d6f59662d7147)

这么设置的好处是可以让不同设备的rem或px都显示一样的长度。

**设置rem**

设置rem的意义在于得到一个与屏幕宽度相关的单位，本来vw是最合适的，但是因为兼容性的问题，只能使用rem来做。这样，让**不同设备的rem显示一样的长度**。

vw是CSS3引入的单位，1vw = 1%窗口宽度

[](https://camo.githubusercontent.com/a15ef368fd6502bfbabb3f2d3174ef9e0694cf9d/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f466f553576674c6f5441504353784e5a5036796c386b675350682d69)

上面的布局我们可以这样：

html{

font-size: 屏幕宽度 / 10;

}

.btn{

width:8.75rem;

height:1.25rem;

}

这样，无论屏幕宽度是多少，.btn都是相对于屏幕的这么宽，做到了适配。

**设置 viewport 缩放 和 data-dpr**

这两个设置其实是干的一件事，就是适配高密度屏幕手机的px单位。

.a{

font-size:12px;

}

[data-dpr="2"] .a{

font-size: 24px;

}

[data-dpr="3"] .a{

font-size: 36px;

}

而缩放是动态的，这样，**不同设备下的px显示一样的长度**。

之前说过CSS像素和物理像素与缩放、dpr都有关系，这里说明：

在普通手机上，.a字体设置为12px；

在dpr是2的手机上，[data-dpr="2"] .a字体为24px，又因为页面缩放50%，字体为还是12px

**总结**

坦白说，我不觉得第一种方案能就做『适配』方案，因为太笨了，只能做一些列表等简单排列的样式。

[](https://camo.githubusercontent.com/25d16b7e19787c9886f2b7e05a249599da5aa3ab/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f4674755736624c4c61306a566d4e445f583074326c6c2d6466316758)

但是一些复杂的活动页的布局，用它可能就力不从心了：

[](https://camo.githubusercontent.com/71d1fad9553132506a721e4ce22bfbe00ec0bcd8/687474703a2f2f37786c6332612e636f6d312e7a302e676c622e636c6f7564646e2e636f6d2f46754c7a2d396d3975755644316b74544b6631686342795432384255)

这是我曾经做过的一个页面，『PK』要和左右两张图平行，而且下面的『不怒自威』、『义薄云天』和下面的战斗力位置都要固定，不能有差。如果用第一种方案，可能各个元素就要绝对定位，然后各种百分比来定位了。且不说计算麻烦，而且辛苦一番最后的结果尺寸是和设计图有出入的。

但是，第二种和第三种方案就绝对不会有这种情况，不会和设计图有差。再说第二种和第三种方案的区别，目前我唯一知道的区别就是第三种方案更加灵活，有两种单位可以使用，想让元素适配的时候就用rem，想让文字不缩放的时候就用px。

如果你没有明白我以上讲的，可能是我太啰嗦了，这是我在团队内分享时做的[PPT](https://riskers.github.io/share/share/flexible.htm)，应该没那么啰嗦了。当然你也可以看看下面的那些文章，不过最好的理解方式就是用这些方案做几个页面，到时候就明白了。