|  |
| --- |
| **开发汇报遇到的问题** |

**1，CSS动画页面闪白,动画卡顿或者有时候 动画效果过多时渲染不出来问题**

答：-webkit-transform:translate3d(0,0,0)触发GPU加速，让网页动画更流畅，transform:translate3d(0,0,0)可以触发硬件加速，这个属性都会开启GPU硬件加速模式，从而让浏览器在渲染动画时从CPU转向GPU。

**2，ios设置overflow-y属性滑动卡顿**

答：给滑动元素设置-webkit-overflow-scrolling:touch。

-webkit-overflow-scrolling 究竟是什么鬼？这是一个只有IOS设备支持的非标准属性，指定是否在 overflow: scroll 的元素中使用“原生”的滚动方式

-webkit-overflow-scrolling: touch; //有回弹效果，原生滚动效果

-webkit-overflow-scrolling: auto; //滑到哪停到哪 普通的无惯性滚动效果

存在的坑：滚动中scrollTop属性不会变化；手势可以穿过其他元素触发滚动效果；运行时通过js动态添加的元素溢出高度导致滚动失效；滚动时暂停其他的transition效果。

**3，输入框获得焦点时，IOS弹出软键盘后整个webView往上弹起无法回弹导致下方出现大片空白且无法拉至顶端**

答：当失去焦点时使用scrollIntoView，使得整个页面再在进入到整个webview

存在的坑：当使用IOS原生键盘时，向上向下键会触发输入框聚焦失焦事件，从而导致页面跳动。

**4，IOS fixed定位抖动问题**

答：（1）fixed定位的元素不与滑动的元素在同一层级，即fixed定位的父元素不存在滚动

（2）使用absolute绝对定位代替fixed

**5，IOS使用new Date解析错误**

答：IOS不支持“2018-12-12”格式的时间，需要转化为“2018/12/12”，可使用'2018-12-12'.replace(/-/g,'/') => '2018/12/12'

**6，移动端Video视频**

答: 6.1 自动播放：autoplay poster属性可放置一个图片链接在视频播放之前展示一个图像

6.2 操作播放：

dom.addEventListener('click', function () { video.play() })

6.3 使用h5播放器，默认打开网页的时候，会自动全屏，不默认全屏播放：webkit-playsinline playsinline x5-playsinlin

6.4部分安卓手机video层级过高：x5-video-player-type="h5"，启用同层H5播放器，就是在视频全屏的时候，div可以呈现在视频层上

**7，IOS 系统中元素被触摸时产生半透明灰色遮罩**

答:把元素的-webkit-tap-highlight-color属性的alpha属性设置为0

**8，问题描述：**

IOS系统，h5页面，当div处于编辑状态时，弹出ios系统自带的软键盘，点击键盘的换行按键，会在光标下方出现空白行，而光标位置不变，软键盘消失后光标位置也不能移动。

**解决思路：**

禁止IOS键盘换行功能，捕获点击换行按钮事件，然后自己处理换行操作

**具体操作：**

<template>

    <div contenteditable="true" @keydown="keydownHandle($event)"></div>

</template>

<script>

export **default**{

    methods:{

        keydownHandle(e) {

*//判断没有按alt键* *且* *按钮为换行键*

**if** (!e.altKey && e.keyCode === 13) {

*//判断系统是否为ios*

**if** (getSys() === 'ios') {

                    e.preventDefault(); *//阻止原生端的默认换行事件*

**let** selection = window.getSelection(); *//* *获取光标的位置*

**let** range = selection.getRangeAt(0); *//* *获取选中的内容*

**if** (!!range.toString()) {

*//* *有选中内容，换行是删除*

                        range.deleteContents();

                    }

**let** br = document.createTextNode('\n');

                    range.insertNode(br); *//* *插入换行*

                    restoreSelection(range, **false**);

                }

            }

        }

    }

}

**function** **restoreSelection**(range, type) {

**if** (typeof type === 'boolean') {

        range.collapse(type); *//将选中的内容光标折叠，参数为true则光标移至选中内容最前端，否则移至尾部*

    }

**let** selection = window.getSelection();

**if** (selection) {

**try** {

            selection.removeAllRanges(); *//* *清空所有Range*

        } **catch** (error) {

            document.body.createTextRange().select();

            document.selection.**empty**();

        }

*//* *将处理好的range对象添加至selection对象中*

        selection.addRange(range)

    }

}

**function** **getSys**(){

**let** u = navigator.userAgent;

**let** isAndroid = u.indexOf('Android') > -1 || u.indexOf('Adr') > -1; *//android终端*

**let** isiOS = /iphone/gi.test(navigator.userAgent); *//ios终端*

**if**(isAndroid){

**return** 'android';

    }

**if**(isiOS){

**return** "ios";

    }

}

</script>

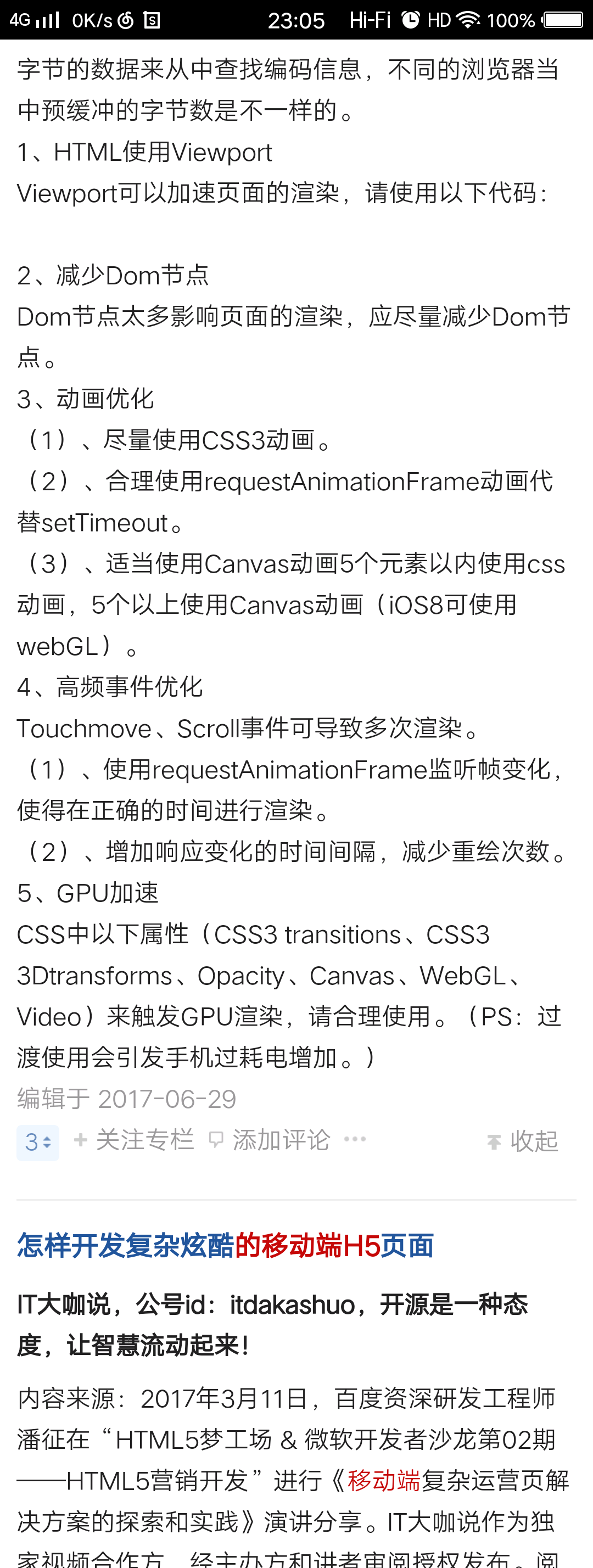
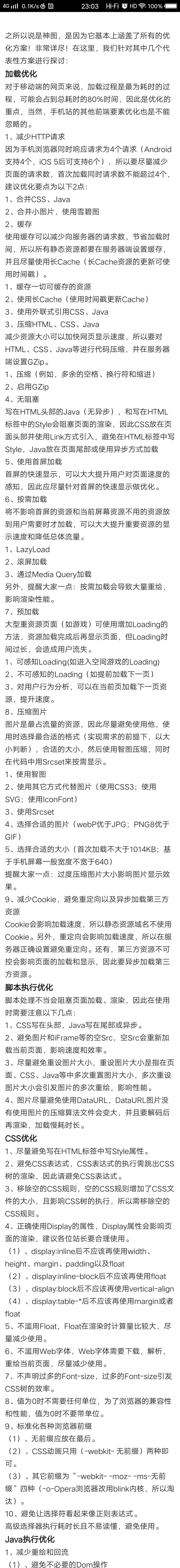
**9，优化方案**

优化无处不在，每一个地方都有可以完善的地方。

首页加载优化：资源加载消耗时间，就算是静态资源的加载也会造成白屏问题的出现。而且，有时候在静态资源加载之前会需要先调接口，这样会导致加载时间更漫长。 据不完全统计，页面出现空白时间超过3秒就会有50%的人想要关闭网页，如果某网站不能在3秒内打开，许多人会失去耐心，转头就走。对于某些购物网站来说，加载时间每延长1秒，一年就会减少数以亿计的营收。这时候，如果在资源加载完成前直接显示一个正在加载中的页面，对用户来说就会完全不一样，用户会觉得网站操作至少不是没有任何反应的。

点击按钮优化：任何操作都会存在一个处理的时间，尤其是需要调接口的操作。这时候如果不给按钮做任何效果，用户可能就会连续点击多次，这样不仅可能造成重复操作，甚至多次频繁调接口可能会导致服务崩溃。所以需要对按钮进行优化，比如操作完成之后再允许点击按钮。而且有时候按钮图标很小，但是需要给按钮区域做的大一点，这样也会改善用户体验。

视觉效果优化：这个在汇报这里遇到了这样一个例子。编辑详情时输入框获得焦点时整个webview会缩小，因为详情的蓝色边框会给用户造成详情页面变小的视觉效果。这个时候我们如果在键盘弹起来的时候去掉底部的边框就会给用户视觉上产生页面并没有变小的效果，对用户体验也会更好一点。在视觉效果优化这方面可以做的很多，比如边框的圆角展示比不用圆角看起来给用户的感觉要好一点，显示柔和一点。比如错误提示，是必须要有的，而且应该是toast形式的，既会用户操作反馈，又不会给用户造成其他操作负担等等。

****

|  |
| --- |
| **移动端适配总结** |

**在 web 的世界里，无线和 PC 的响应式适配其实是两个世界……**

**1. 视口 viewport**

**1.1 viewport 基础**

viewport 解释为中文就是‘视口’的意思，也就是浏览器中用于显示网页的区域。在 PC 端，其大小也就是浏览器可视区域的大小，所以我们也不会太关注此概念；而在移动端，绝大多数情况下 viewport 都大于浏览器可视区，保证 PC 页面在移动浏览器上面的可视性。为提升可视性体验，针对移动端有了对 viewport 的深入研究。

**1.2 viewport 详解**

在移动端有三种类型的 viewport: layoutviewport、visualviewport、idealviewport。具体解释如下：

* layoutviewport: 大于实际屏幕， 元素的宽度继承于 layoutviewport，用于保证网站的外观特性与桌面浏览器一样。layoutviewport 到底多宽，每个浏览器不同。iPhone 的 safari 为 980px，通过 document.document.clientWidth 获取。
* visualviewport: 当前显示在屏幕上的页面，即浏览器可视区域的宽度。
* idealviewport: 为浏览器定义的可完美适配移动端的理想 viewport，固定不变，可以认为是设备视口宽度。比如 iphone 7 为 375px, iphone 7p 为 414px。

**1.3 viewport 设置**

我们通过对几种 viewport 设置可以对网页的展示进行有效的控制，在移动端我们经常会在 head 标签中看到这段代码：

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

通过对 meta 标签三个 viewport 的设置，最终使页面完美展示。下面详细的阐释其具体含义:

* width 设置的是 layoutviewport 的宽度
* initial-scale 设置页面的初始缩放值，并且这个初始缩放值是相对于 idealviewport 缩放的，最终得到的结果不仅会决定 visualviewport，还会影响到 layoutviewport
* user-scalable 是否允许用户进行缩放的设置

对上面的说明通过公式推导进行进一步的解释：

// 设定两个变量：

viewport\_1 = width;

viewport\_2 = idealviewport / initial-scale;

// 则：

layoutviewport = max{viewport\_1, viewport\_2};

visualviewport = viewport\_2;

结合上面对不同 viewport 分析补充如下两个结论：

1. 若 layoutviewport === idealviewport && visualviewport === idealviewport，则三个值一致时，页面会完美展示。
2. 若 layoutviewport === visualviewport 且它们不等于 visualviewport，页面下面不会出现滚动条，浏览器默认只是把页面放大或缩小。

**1.4 viewport 举例**

以下是通过改变 meta viewport 的几个参数的值来算取不同的 viewport：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| width | initial-scale | layoutviewport | visualviewport | idealviewport | 是否滚动 |
| - | - | 980px | 980px | 375px | 否 |
| device-width | 1 | 375px | 375px | 375px | 否 |
| device-width | 2 | 375px | 188px | 375px | 是 |
| device-width | 0.5 | 750px | 750px | 375px | 否 |
| 480px | 1 | 480px | 375px | 375px | 是 |
| 480px | 2 | 480px | 188px | 375px | 是 |
| 480px | 0.5 | 750px | 750px | 375px | 否 |

以上是针对 iphone 6/7/8 的测试数据，且无论怎么设置 viewport 都具有临界值，即：75 <= layoutviewport <= 10000，75 <= visualviewport <= 1500。

**1.5 为什么要设置 viewport**

viewport 的设置不会对 PC 页面产生影响，但对于移动页面却很重要。下面我们举例来说明：

1. 媒体查询 @media 响应式布局中，会根据媒体查询功能来适配多端布局，必须对 viewport 进行设置，否则根据查询到的尺寸无法正确匹配视觉宽度而导致布局混乱。如不设置 viewport 参数，多说移动端媒体查询的结果将是 980px 这个节点布局的参数，而非我们通常设置的 768px 范围内的这个布局参数
2. 由于目前多数手机的 dpr 都不再是 1，为了产出高保真页面，我们一般会给出 750px 的设计稿，那么就需要通过设置 viewport 的参数来进行整体换算，而不是在每次设置尺寸时进行长度的换算。

**2. 设备像素比 dpr 与 1px 物理像素**

**2.1 物理像素（physical pixel）**

手机屏幕上显示的最小单元，该最小单元具有颜色及亮度的属性可供设置，iphone6、7、8 为：750 \* 1334，iphone6+、7+、8+ 为 1242 \* 2208

**2.2 设备独立像素（density-indenpendent pixel）**

此为逻辑像素，计算机设备中的一个点，css 中设置的像素指的就是该像素。老早在没有 retina 屏之前，设备独立像素与物理像素是相等的。

**2.3 设备像素比（device pixel ratio）**

设备像素比(dpr) = 物理像素/设备独立像素。如 iphone 6、7、8 的 dpr 为 2，那么一个设备独立像素便为 4 个物理像素，因此在 css 上设置的 1px 在其屏幕上占据的是 2个物理像素，0.5px 对应的才是其所能展示的最小单位。这就是 1px 在 retina 屏上变粗的原因，目前有很多办法来解决这一问题。

**2.4 1px的物理像素的解决方案**

从第一部分的讨论可知 viewport 的 initial-scale 具有缩放页面的效果。对于 dpr=2 的屏幕，1px压缩一半便可与1px的设备像素比匹配，这就可以通过将缩放比 initial-scale 设置为 0.5=1/2 而实现。以此类推 dpr=3的屏幕可以将 initial-scale设置为 0.33=1/3 来实现。

**3. 设备像素比 dpr 与 rem 的适配方案**

结合 2、3 部分可以实现 1px 的物理像素这一最小屏幕单位，那在此基础上如可让设计通常提供的 750px 设计稿来完美的适配到多种机型上，使用 rem 是一种解决方式。

**3.1 rem 如何设置**

rem 是相对于根元素 html 的 font-size 来做计算。通常在页面初始化时加载时通过对document.documentElement.style.fontSize 设置来实现。

**3.2 rem 适配规则**

通过对 initial-scale = 1/dpr 的设置，已将对屏幕的描述从物理像素转化到了物理像素上了，这将是后续推导的基础，且设计稿为 750px。

1. 物理像素为 750 = 375 \* 2，若屏幕等分为 10 份，那么 1rem = 75px，10rem = 750px;
2. 物理像素为 1125 = 375 \* 3，若屏幕等分为 10 份，那么 1rem = 112.5px, 10rem = 1125px;
3. 物理像素为 1242 = 414 \* 3, 若屏幕等分为 10 份，那么 1rem = 124.2px, 10rem = 1242px;

因此可推导出 rem 的设定方式：

document.documentElement.style.fontSize = document.documentElement.clientWidth / 10 + 'px';

下面我们将 750px 下，1rem 代表的像素值用 baseFont 表示，则在 baseFont = 75 的情况下，是分成 10 等份的。因此可以将上面的公式通用话一些：

document.documentElement.style.fontSize = document.documentElement.clientWidth / ( 750 / 75 ) + 'px';

整体设置可参考如下代码：

(**function** (baseFontSize) {

**const** \_baseFontSize = baseFontSize || 75;

**const** ua = navigator.userAgent;

**const** matches = ua.match(/Android[\S\s]+AppleWebkit\/(\d{3})/i);

**const** isIos = navigator.appVersion.match(/(iphone|ipad|ipod)/gi);

**const** dpr = window.devicePixelRatio || 1;

**if** (!isIos && !(matches && matches[1] > 534)) {

        // 如果非iOS, 非Android4.3以上, dpr设为1;

        dpr = 1;

    }

**const** scale = 1 / dpr;

**const** metaEl = document.querySelector('meta[name="viewport"]');

**if** (!metaEl) {

        metaEl = document.createElement('meta');

        metaEl.setAttribute('name', 'viewport');

        window.document.head.appendChild(metaEl);

    }

    metaEl.setAttribute('content', 'width=device-width,user-scalable=no,initial-scale=' + scale + ',maximum-scale=' + scale + ',minimum-scale=' + scale);

    document.documentElement.style.fontSize = document.documentElement.clientWidth / (750 / \_baseFontSize) + 'px';

})();

同时为了书写方便可以直接通过 px 布局，然后在打包时利用 pxtorem 库转化为基于 rem 的布局。

**4. 视口单位适配方案**

将视口宽度 window.innerWidth 和视口高度 window.innerHeight 等分为 100 份，且将这里的视口理解成 idealviewport 更为贴切，并不会随着 viewport 的不同设置而改变。

* vw : 1vw 为视口宽度的 1%
* vh : 1vh 为视口高度的 1%
* vmin : vw 和 vh 中的较小值
* vmax : 选取 vw 和 vh 中的较大值

如果设计稿为 750px，那么 1vw = 7.5px，100vw = 750px。其实设计稿按照设么都没多大关系，最终转化过来的都是相对单位，上面讲的 rem 也是对它的模拟。这里的比例关系也推荐不要自己换算，使用 pxtoviewport 的库就可以帮我们转换。当然每种方案都会有其弊端，这里就不展开讨论。

**总结**

在移动端开发中，理解视口对适配至关重要。因此本文先从视口展开讨论，从而引出 1px、rem 及 vw/vh 这些和适配相关的主要话题。下面提供的参考文章会在某些点上更加细化，以供参考。