移动端H5最佳实践

REM单位布局适配方案

在页面开发中，`rem`（Root EM）是一种相对单位，用于在网页设计和开发中定义元素的尺寸和间距。它基于HTML根元素（通常是`<html>`）的字体大小，而不是直接依赖于父元素的字体大小，比如，如果根元素的字体大小是16px，那么1rem就等于16px。

这使得REM在响应式设计中非常有用，因为所有的尺寸都是基于一个统一的参考点（即根元素的字体大小），只需修改根元素的字体大小，整个页面的比例会自动调整。

常规方案

通常移动端设计稿的尺寸为`750px`

在代码中直接使用`rem`进行样式的开发，需要进行以下2个步骤

实现过程

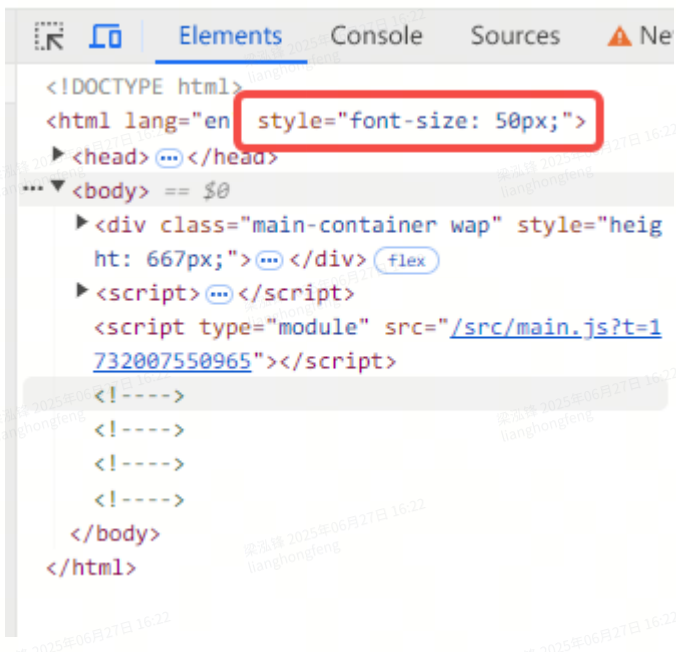
1. 初始化根元素字体大小

通过JavaScript读取屏幕宽度，然后根据宽度计算出对应的尺寸并设置根元素的`font-size`。

以下是需要调整至1rem=100px的初始化代码。

```
// 确定设计稿宽度和基准值，即屏幕物理像素和设计稿一致的情况下，根元素font-size设置成多少
const designWidth = 750;
const remBase = 50;
// 根据当前屏幕宽度得到最终的根元素font-size
const rootFontSize = window.innerWidth / designWidth * remBase;
// 更新根元素的font-size
document.documentElement.style.fontSize = rootFontSize + "px";
```

设置完成后，浏览器中的font-size在iphone6s（宽度375px）中计算结果为50px。



> 根元素的font-size不宜过小，这是由于部分旧机型和浏览器不支持小于12px的字体，小于12px仍按12px去计算，从而导致实际尺寸偏大的问题。

评论（1）

> 屏幕物理像素和设计稿一致

刘滨 2024年11月19日 （编辑过）

这一标准是 iphone6s 时期约定的，那时候大部分设备 dpr 都是 2，沿用至今，直接按 2 倍计算即可。

iphone6s
物理像素：750px
实际像素：375px
设计稿尺寸：750px

因此 1:100 的换算比例下计算出来的根元素 font-size 是 50px

2. 设计稿单位换算

基于第1步的设置，我们可以知道在写尺寸单位的时候，需要拿设计稿尺寸除以 `100` 再加上 `rem` 。

```
// 设计稿尺寸
.logo {
    padding: 24px;
}

// 开发实际编写尺寸
.logo {
    padding: 0.24rem;
}
```

不足之处

1. 写样式需要自行换算数值，相对直接写 `px` 而言不是那么直观
2. 第三方组件（ `vant` ）、迁移代码（从其它项目复制过来的组件）中的样式文件不方便直接改造，导致出现 `rem` + `px` 混合使用的情况
3. 所有样式写死了 `rem` 单位，如果将来有更好的样式方案（如目前同样流行的vw），或者样式需要迁移到小程序（小程序单位需要使用 `rpx` ，不支持 `rem` ），由于所有的尺寸数值必须改变，不好改造

改良措施

1. 通过编译将px转换为rem

基于常规方案的不足之处，我们希望通过编译解决数值换算的问题，在开发过程中使用和设计稿一致的数值+ `px` 来写代码，在编译产物中全部转换成 `rem` 尺寸。

借助postcss-pxtorem可以做到，这是 `postcss` 的插件，和打包工具(`vite` / `webpack`)无关。

基于常规方案第1步的设置，编译后代码1rem=设计稿100px，计算出来的font-size是50px，进行如下配置

```
// main.js
// 确定设计稿宽度和基准值，即屏幕物理像素和设计稿一致的情况下，根元素font-size设置成多少
const designWidth = 750; // 设计稿尺寸基准
const remBase = 50; // 根元素font-size，由于实际宽度缩小一倍，算出来是25
// 根据当前屏幕宽度得到最终的根元素font-size
const rootFontSize = window.innerWidth / designWidth * remBase;
// 更新根元素的font-size
document.documentElement.style.fontSize = rootFontSize + "px";
```

```
// vite: vite.config.js属性链路为css -> postcss
// webpack: 到postcss-loader下配置
export default {
    // ...
    postcss: {
       plugins: [
          pxtorem({
```

```
                rootValue: 50, // 设置模拟器为iphone6s，此时计算出来根元素的font-size为50px，填到这里
                mediaQuery: false,
                propList: ["*"],
            }),
        ],
    }
}
```

配置完成后，尺寸单位即可按照设计稿的数值写px进行开发，此外，所有经过入口导入和编译的第三方包也会统一编译为 `rem` 单位。

2. 注意统一第三方组件和页面的尺寸数值和单位

注意下面这张表，如果设计稿尺寸基准是750px，插件的rootValue设为50，经过转换后屏幕中的实际像素是代码的2倍

方案	设计稿宽度	根元素 font-size	375px转换后数值 / 实际像素	100px转换后数值 / 实际像素	1px转换后数值 / 实际像素
2倍屏设计稿	**750px**	**25px**	7.5rem / 187.5px	2rem / 50px	0.02rem / 0.5px
实际像素	**375px**	**100px**	3.75rem / 375px	1rem / 100px	0.01rem / 1px

这就带来一个问题，有些第三方组件例如 `vant@2` 是 `px` 单位写尺寸的，如果根元素按 `50px` ，它的所有元素渲染出来的实际尺寸都会变为原来的一半，达不到预期效果。如果按 `100px` 去转换，则没有这个问题。

因此，面对这种情况，需要事先考虑好，设计稿的尺寸基准应该选择750还是375。

最终解决方案：可以将vant的样式按另一个基准去转换

```
module.exports = {
  plugins: {
    'postcss-pxtorem': {
      rootValue({ file }) {
        return file.indexOf('vant') !== -1 ? 100 : 50;
      },
      propList: ['*'],
    },
  },
};
```

输入框交互体验

手机号和验证码输入框拉起数字键盘

```
<input type="tel" pattern="[0-9]*" max-length="11" />
<input type="number" pattern="[0-9]*" max-length="6" />
```

type="tel"

type="number"

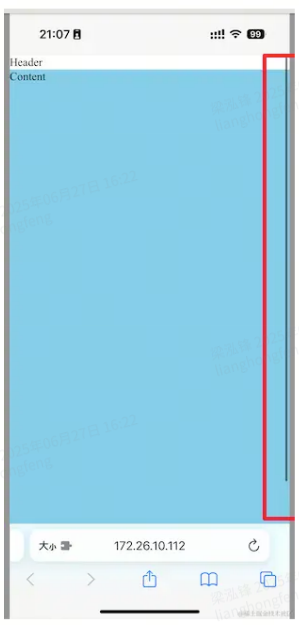
短信验证码自动填充

> 需要手机已授权开启了自动识别短信验证码功能，一般都是默认开启的
>
> 必须设置maxlength，否则可能会出现验证码重复填充的问题

```
<input type="number" maxlength="6" autocomplete="one-time-code" />
```

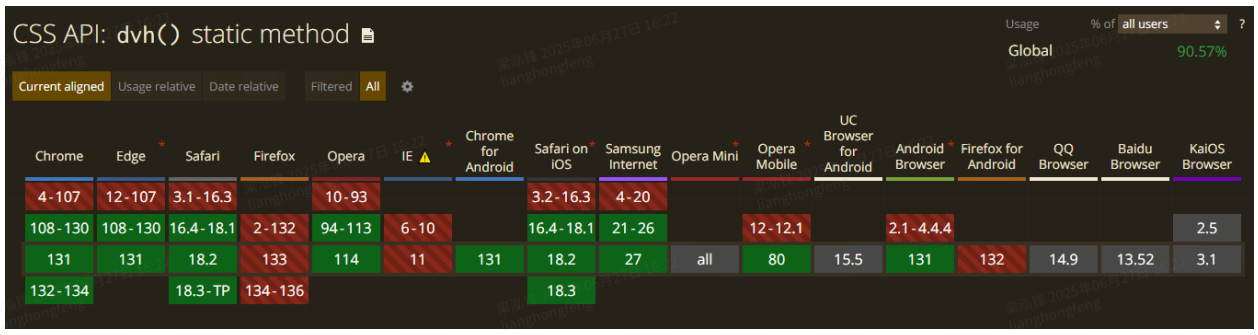
动态视口适配

在IOS safari浏览器中，导航栏和工具栏是悬浮在页面上方的，这导致当我们想设置一个元素的高度为100vh时（通常用于做局部滚动），这个元素会比页面的视口（innerWidth）更高，导致出现滚动条。



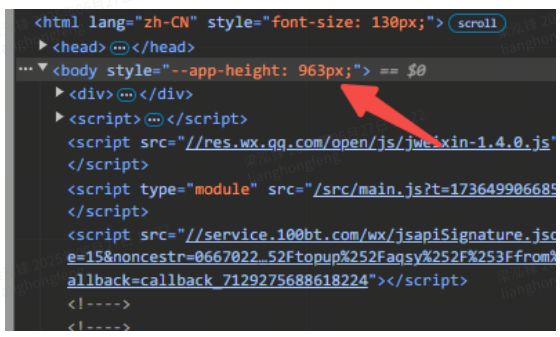
对此，w3c推出了新的单位：dvh（动态视口高度），实际上就是动态的window.innerHeight

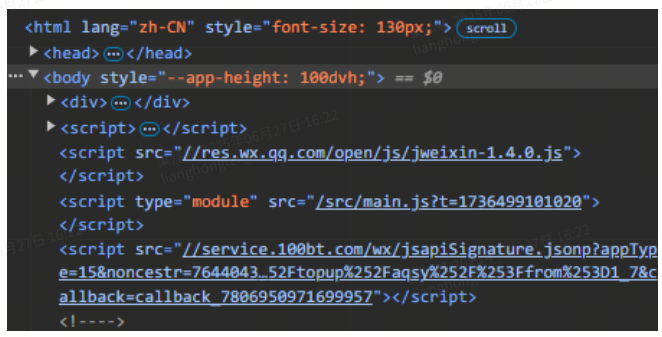
最佳的解决方案是使用100dvh代替100vh，但是它对旧机型的兼容性不好



但可以借助JS辅助实现类似的效果，代码如下

```
document.body.style.setProperty("--app-height", "100dvh");
if (!CSS.supports("height", "100dvh")) {
  // 如果不支持，就定义 --app-height 为视口高度，即 window.innerHeight
  document.body.style.setProperty("--app-height", window.innerHeight + "px");
  // 当屏幕缩放时，改变内容高度。因为 resize 事件触发很频繁，所以使用节流减少性能损耗
  let timeout;
  function onDynamicHeightResize() {
    clearTimeout(timeout);
    timeout = setTimeout(() => {
      clearTimeout(timeout);
      document.body.style.setProperty(
        "--app-height",
        window.innerHeight + "px"
      );
    }, 500);
  }
  window.addEventListener("resize", onDynamicHeightResize);
}
```





在CSS中，只需要这么写即可

```
.conainer {
    height: var(--app-height);
}
```

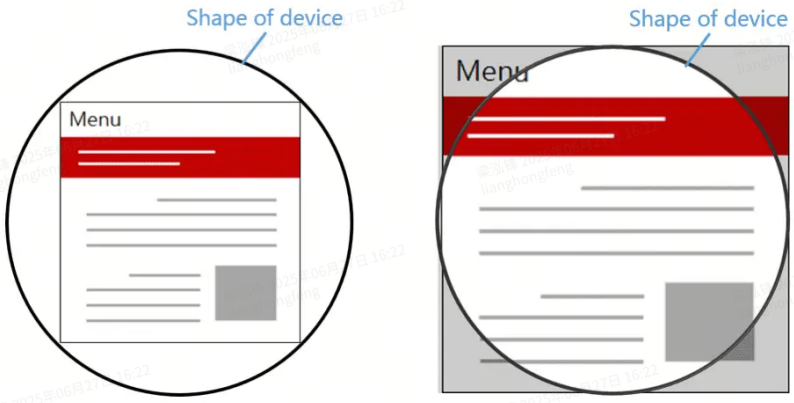
iPhoneX底部安全区适配

1. html头部设置viewport-fit=cover

viewport-fit的3个值：

- contain：可视窗口完全包含网页内容（左图）
- cover：网页内容完全覆盖可视窗口（右图）✅

- auto：默认值，此值不影响初始布局视图端口，并且整个web页面都是可查看的。




```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, viewport-fit=cover" />
```

2. 在需要预留安全区的元素设置CSS

> 注意顺序不能错

可以添加paading-bottom，或者放一个空白的 `<div class="safe-area">` 元素

```
.safe-area {
    @supports (bottom: constant(safe-area-inset-bottom)) or (bottom: env(safe-area-inset-bottom)) {
        height: constant(safe-area-inset-bottom); /*兼容 IOS<11.2*/
        height: env(safe-area-inset-bottom); /*兼容 IOS>11.2*/
    }
}
```

最终效果

无微信导航栏

有微信导航栏

浏览器

微信动态导航栏标题设置

微信访问H5时，无法直接通过`document.title`动态更改导航栏的标题，通常借助iframe来更新，封装方法如下

```
const setIosWechatTitle = (title) => {
  document.title = title;
  const iframe = document.createElement("iframe");
  iframe.src = "/";
  iframe.style.display = "none";

  iframe.onload = function () {
    setTimeout(function () {
      iframe.remove();
    }, 0);
  };
  document.body.appendChild(iframe);
};

export function usePageTitle(title) {
  if (isIOS) {
    setIosWechatTitle(title);
  } else {
    document.title = title;
  }
}

```

禁用横屏提示

1. 根组件放置代码

```
<div class="wap-orient-tips">
  <div class="inner">
    <img src="https://res-static.172tt.com/common/img/phone-rotate.png" />
    <span class="desc"></span>
  </div>
</div>
```

2. 引用CSS

```css
.wap-orient-tips {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  z-index: 9999;
  background: rgba(0, 0, 0, 0.9);
  flex-direction: column;
  align-items: center;
  justify-content: center;
  display: none;
  color: #fff;
}
.wap-orient-tips .inner {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.wap-orient-tips .inner::after {
  content: "为了更好的体验，请将手机/平板竖过来";
  font-size: 10px;
}
.wap-orient-tips img {
  width: 60px;
  height: 60px;
  margin-bottom: 4px;
  object-fit: contain;
  animation: rotateAni 1.5s ease infinite alternate;
}
.wap-orient-tips span {
  font-size: 10px;
}
@media screen and (min-aspect-ratio: 4 / 3) and (max-width: 992px) {
  .wap-orient-tips {
    display: flex;
  }
}
@keyframes rotateAni {
  0% {
    transform: rotate(-90deg);
  }
  30% {
    transform: rotate(-90deg);
  }
  70% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(0deg);
  }
}

```

已封装为snowball组件，直接引用js即可

https://snowball.172tt.com/wap-orient-tips/README.html

阻止页面缩放

页面缩放有2种方式触发：

- 双指捏合缩放
- 双击屏幕放大

以前我们可以通过meta阻止缩放：

代码块

```
<meta
  name="viewport"
  content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, viewport-fit=cover"
/>
```

但是在IOS 10以上的版本，这行代码已经不管用了，需要通过JS对上述2点手势操作进行处理

代码块

```
// 处理IOS 10以上可双指放大和可双击放大的问题
window.onload = function () {
  // 阻止双击放大
  let lastTouchEnd = 0;
  document.addEventListener("touchstart", function (event) {
    if (event.touches.length > 1) {
      event.preventDefault();
    }
  });
  document.addEventListener(
    "touchend",
    function (event) {
      const now = new Date().getTime();
      if (now - lastTouchEnd <= 300) {
        event.preventDefault();
      }
      lastTouchEnd = now;
    },
    false
  );
  // 阻止双指放大
  document.addEventListener("gesturestart", function (event) {
    event.preventDefault();
  });
};

```

点击反馈

如果页面对动效有一定要求，通常需要加入一些点击反馈让操作更加跟手，可以通过伪类实现

代码块

```
// 点击时出现阴影遮罩
.click--highlight {
  position: relative;
  overflow: hidden;

  &:after {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.3);
    z-index: 100;
    opacity: 0;
    transition: opacity 0.2s ease-in-out;
  }

  &:active:after {
    opacity: 1;
  }
}

// 点击时元素变小
.click--smaller {
  transition: transform 0.1s linear;

  &:active {
    transform: scale(0.9);
  }
}
```

在PC模拟器下我们可以看到:active伪类的效果，但是在IOS下仍然未生效，原因是IOS会先判定touchstart事件（长按的情况），从而让active伪类没有立即生效，因此需要通过JS阻止touchstart的判定，增加以下代码即可解决

代码块

```

// 让css的:active伪类在移动端下也生效
document.addEventListener(
  "touchstart",
  function () {
    return false;
  },
  false
);
```

自动抓取图片和字体资源进行预加载

原理：

1. 提前将需要渲染的资源通过JS请求一遍（期间显示loading动画），在浏览器二次加载时会命中缓存从而免去二次网络请求
2. window.onload时通过抓取head中StyleSheet的内容，可将所有匹配到后缀的资源收集起来进行预加载

好处：避免用户在使用过程中因资源未加载完成导致的画面抖动

执行以下方法即可进行资源的预加载（非阻塞，加载失败仍然按成功处理）

代码块

```
preloadAssets().then(() => {
    new Vue({ render: h => h(App) })
})
```

源码如下：

代码块

```
// 预加载图片
const preloadImage = (url: string): Promise<void> => {
  return new Promise((resolve, reject) => {
    const img = new Image();
    img.onload = () => resolve();
    img.onerror = reject;
    img.src = url;
  });
};

// 预加载字体
const preloadFont = (url: string): Promise<void> => {
  return new Promise((resolve, reject) => {
    const font = new FontFace("preload-font", `url(${url})`);
    font
      .load()
      .then(() => resolve())
      .catch(reject);
  });
};

// 从CSS中提取资源URL
const extractUrlsFromCSS = (cssText: string): { images: string[]; fonts: string[] } => {
  const images: string[] = [];
  const fonts: string[] = [];

  // 匹配url()中的图片
  const imageRegex = /url\(['"]?([^'"()]+\.(?:png|jpg|jpeg|gif|webp|svg))['"]?\)/gi;
  let match;
  while ((match = imageRegex.exec(cssText)) !== null) {
    images.push(match[1]);
  }

  // 匹配@font-face中的字体
```

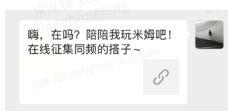
```
  const fontRegex = /@font-face\s*{[^}]*url\(['"]?([^'"()]+\.(?:woff2?|ttf|eot|otf))['"]?\)[^}]*}/gi;
  while ((match = fontRegex.exec(cssText)) !== null) {
    fonts.push(match[1]);
  }

  return { images, fonts };
};

// 主预加载函数
export const preloadAssets = async (): Promise<boolean> => {
  // 获取所有样式表
  const styleSheets = Array.from(document.styleSheets);
  console.log({ styleSheets });
  let cssText = "";

  // 收集所有CSS文本
  styleSheets.forEach(sheet => {
    try {
      const rules = Array.from(sheet.cssRules);
      rules.forEach(rule => {
        cssText += rule.cssText;
      });
    } catch (e) {
      console.warn("无法读取样式表:", e);
    }
  });

  // 提取资源URL
  const { images, fonts } = extractUrlsFromCSS(cssText);

  console.log(cssText);

  console.log({
    cssText,
    images,
    fonts
  });

  const taskLength = images.length + fonts.length;

  let progress = 0;
  const preloadTask = async (task: () => Promise<void>) => {
    try {
      await task();
      console.log("资源加载中", `${progress + 1}/${taskLength}`);
      progress++;
    } catch (err) {
      console.error("预加载资源失败:", err);
    }
    return true;
  };

  // 预加载所有资源
  const preloadPromises = [
    ...images.map(url => preloadTask(() => preloadImage(url))),
    ...fonts.map(url => preloadTask(() => preloadFont(url)))
  ];

  await Promise.all(preloadPromises);
  console.log("资源加载完成");

```

```
96       return true;
97     };
98
```

由于它只抓取CSS中的资源，因此需要注意以下2点

1. 需要在CSS中引入资源，html中引入的获取不到
2. 对于vite打包的项目，在vue SFC中写的CSS，为了按需加载做了拆包，项目初始化时可能获取不到，因此建议写成单入口的形式，统一通过 `/src/styles/index.less` 来引入

分享信息优化

环境	微信		QQ	浏览器（分享到微信/QQ）
	公众号网页	非公众号网页		
优化方式	对接微信jssdk	无，只支持获取title	IOS： 1. 标题和描述动态获取当前页面的<title>, <meta name="description ...> 2. 封面图获取当前页面DOM中的第一个img元素src Android： • 分享时远程获取线上url的TDK，不支持动态设置 • 线上url获取不到时，用标题：“分享”、封面图：“当前截图”和描述：url兜底	获取当前页面的TDK，建议按照Open Graph（开放图谱）协议 配置 <meta name="og:title"> <meta name="og:description"> <meta name="og:image">
是否支持运行时动态更改	支持	不支持	IOS支持（但可能有缓存） Android不支持	支持
分享效果			 IOS  Android线上地址不存在  安卓线上地址存在且设置了TDK	 

结论：

1. 针对IOS端QQ分享的场景，建议页面中设置第一个img标签为封面图，样式设置为隐藏，避免客户端抓取到无关的图片当封面
2. 针对Android端QQ分享的场景，需要增加<meta name="og:image">（第3点有提到）
3. 我们以前常常忽略了从浏览器分享到App的场景，建议优化按照Open Graph协议增加<meta name="og:title"><meta name="og:description"><meta name="og:image">等信息