# 面试官：你知道移动端适配吗?

## 自适应和响应式

- 自适应：根据不同的设备屏幕大小来自动调整尺寸、大小
- 响应式：会随着屏幕的实时变动而自动调整，是一种更强的自适应

## 为什么要做移动端适配?

目前市面上移动端屏幕尺寸非常的繁多，很多时候我们希望一个元素在不同的屏幕上显示不同的大小以此来更好的还原效果图：

> - 当我们针对一个手机进行布局设计时，设置了一个 200 * 200 大小的盒子
> - 但在不同的设备上，由于逻辑像素不同的问题，会使得这个盒子在更大的视口上显得很小，在更小的视口上显示很大
> - 因此我们需要进行适配**让它在不同设备上所占据视口的空间比例是相同的**

## 当前流行的几种适配方案

- 方案一：百分比设置（不推荐）
  - 因为不同属性的百分比值，相对的可能是不同参照物，所以百分比往往很难统一
  - 所以百分比在移动端适配中使用是非常少的
- 方案二：rem单位+动态html的font-size
- 方案三：vw单位（推荐）
- 方案四：flex的弹性布局

## rem + 动态设置 font-size

rem 单位是相对于 html 元素的 font-size 来设置的，通过在不同屏幕尺寸下，动态的修改 html 元素的 font-size 以此来达到适配效果

在开发中，我们只需要考虑两个问题：

- 针对不同的屏幕，设置 html 不同的 font-size
- 将原来设置的尺寸，转化成 rem 单位

**font-size 的尺寸**

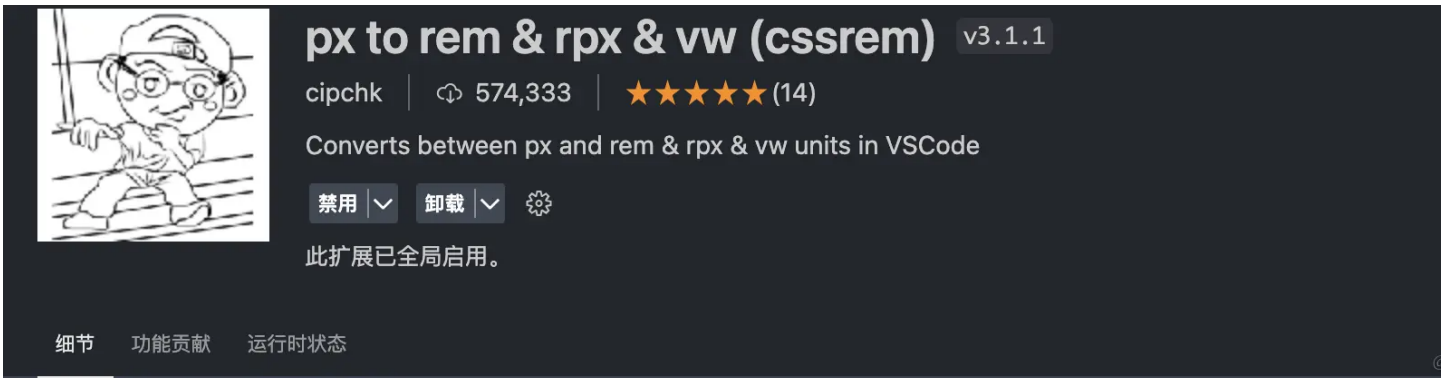
| 屏幕尺寸 | html的font-size | 盒子的设置宽度 | 盒子的最终宽度 |
| --- | --- | --- | --- |
| 375px | 37.5px | 1rem | 37.5px |
| 320px | 32px | 1rem | 32px |
| 414px | 41.4px | 1rem | 41.4px |

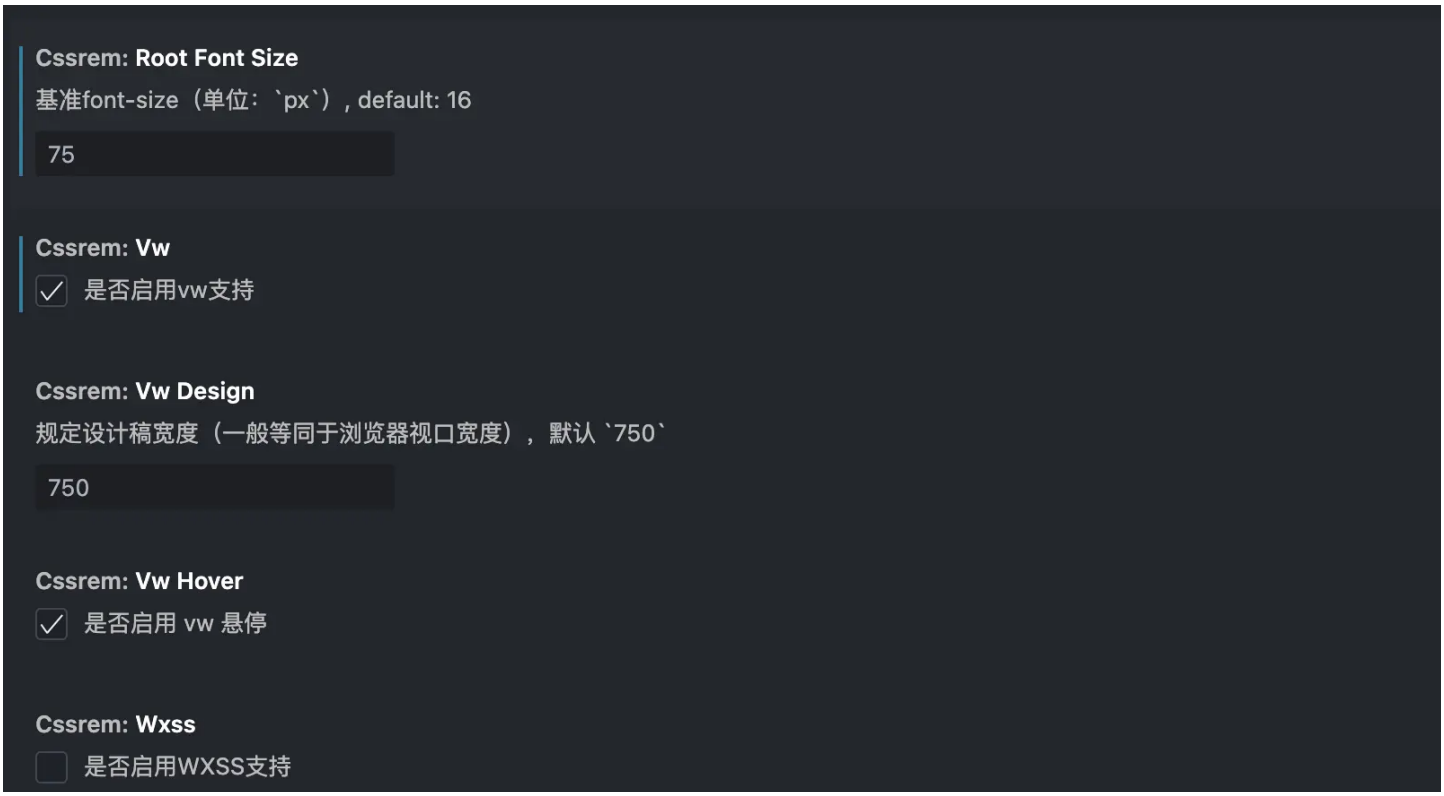
## px 与 rem 的单位换算

- **手动换算**
  - 根元素 html 的文字大小 = 视口宽度/分成的份数(一般为10份，方便计算)
  - rem 值 = 元素的 px 值 / 根元素 html 的文字大小
  - 比如有一个在375px屏幕上，100px宽度和高度的盒子
  - 我们需要将100px转成对应的rem值
  - 100/37.5=2.6667，其他也是相同的方法计算即可
- **less/scss函数**

```
.pxToRem(@px) {
  result: 1rem * (@px / 37.5);
}

.box {
  width: .pxToRem(100)[result];
  height: .pxToRem(100)[result];
  background-color: orange;
}

p {
  font-size: .pxToRem(14)[result];
}


```

- **postcss-pxtorem**
  - 目前在前端的工程化开发中，我们可以借助于webpack的工具来完成自动的转化
  - `npm install postcss-pxtorem`
- **VSCode插件**




## 方案一：媒体查询

思路：通过媒体查询来设置不同尺寸屏幕下 html 的 font-size

缺点：

- 需要针对不同的屏幕编写大量的媒体查询
- 如果动态改变尺寸，不会实时更新，只是一个个区间

```
@media screen and (min-width: 320px) {
    html {
      font-size: 20px;
    }
}

@media screen and (min-width: 375px) {
   html {
     font-size: 24px;
   }
```

```
}

@media screen and (min-width: 414px) {
  html {
    font-size: 28px;
  }
}

@media screen and (min-width: 480px) {
  html {
    font-size: 32px;
  }
}

.box {
  width: 5rem;
  height: 5rem;
  background-color: blue;
}
```

## 方案二：编写 js 代码

思路：通过监听屏幕尺寸的变化来动态修改 html 元素的 font-size 大小

方法：

- 根据 html 的宽度计算出 font-size 的大小，并设置到 html 上
- 监听页面尺寸的变化，实时修改 font-size 大小

```
function setRemUnit() {
  // 获取所有的 html 元素
  const htmlEl = document.documentElement
  // 375 -> 16px
  // 320px -> 12px
  // 我们需要动态更改字体大小，因此需要获取网页的宽度
  // 拿到客户端宽度
  const htmlWidth = htmlEl.clientWidth
  // 将宽度分成10份
  const htmlFontSize = htmlWidth / 10
  console.log('htmlFontSize', htmlFontSize);
  // 将值给到html的font-size
  htmlEl.style.fontSize = htmlFontSize + 'px'
}

```

```
setRemUnit()
// 给 window 添加监听事件
window.addEventListener('resize', setRemUnit)
// 375 -> 16px  // 320px -> 12px  // 我们需要动态更改字体大小，因此需要获取网页的宽度  // 拿到客户端宽度  const htmlWidth = htmlEl.clientWidth  // 将宽度分成10份  const htmlFontSize = htmlWidth / 10  console.log('htmlFontSize', htmlFontSize);  // 将值给到html的font-size  htmlEl.style.fontSize = htmlFontSize + 'px'}setRemUnit()// 给 window 添加监听事件window.addEventListener('resize', setRemUnit)
```

## 方案三：lib-flexible 库

lib-flexible 是淘宝团队出品的一个移动端自适应解决方案，通过动态计算 viewport 设置 font-size 实现不同屏幕宽度下的 UI 自适应缩放。

```
(function flexible (window, document) {
  var docEl = document.documentElement
  var dpr = window.devicePixelRatio || 1

  // adjust body font size
  function setBodyFontSize () {
    if (document.body) {
      document.body.style.fontSize = (12 * dpr) + 'px'
    }
    else {
      document.addEventListener('DOMContentLoaded', setBodyFontSize)
    }
  }
  setBodyFontSize();

  // set 1rem = viewWidth / 10
  function setRemUnit () {
    var rem = docEl.clientWidth / 10
    docEl.style.fontSize = rem + 'px'
  }

  setRemUnit()

  // reset rem unit on page resize
  window.addEventListener('resize', setRemUnit)
  window.addEventListener('pageshow', function (e) {
    if (e.persisted) {
      setRemUnit()
    }
  })
```

```
31
32    // detect 0.5px supports
33    if (dpr >= 2) {
34      var fakeBody = document.createElement('body')
35      var testElement = document.createElement('div')
36      testElement.style.border = '.5px solid transparent'
37      fakeBody.appendChild(testElement)
38      docEl.appendChild(fakeBody)
39      if (testElement.offsetHeight === 1) {
40        docEl.classList.add('hairlines')
41      }
42      docEl.removeChild(fakeBody)
43    }
44  }(window, document))
45
```

## vw 适配方案

100vw 相当于整个视口的宽度 innerWidth，1vw 相当于视口宽度的 1%，将 px 转换为 vw 即可完成适配，其实上面的 rem 就是模仿 vw 方案

**vw相比于rem的优势：**

- 不需要去计算 html 的 font-size 大小，也不需要去给 html 设置 font-size
- 不会因为设置 html 的 font-size 大小，而必须再给 body 设置一个 font-size 防止继承
- 因为不依赖 font-size 的尺寸，所以不用担心某些原因的 html 的 font-size 尺寸被篡改，导致页面尺寸混乱
- vw 更加语义话，1vw 相当于 1/100 viewport 的大小
- rem 事实上作为一种过渡的方案，它利用的也是 vw 的思想

## px 与 vw 的单位转换

**手动换算**

比如屏幕尺寸为 375px，元素大小为 150px，我们需要将 150px 转换成对应的 vw 值：`150 / 3.75=40`

**less/scss 函数**

```
1 scss
2 复制代码
3 @vwUnit: 3.75;.pxToVw(@px) {  result: (@px / @vw) * 1vw}.box {  width:
  .pxToVw(100)[result];  height: .pxToVw(100)[result];}
```

### postcss-px-to-viewport-8-plugin

- 和rem一样，在前端的工程化开发中，我们可以借助于webpack的工具来完成自动的转化
- `npm install postcss-px-to-viewport-8-plugin`

### vs Code 插件

px to vw 插件，在编写时自动转化：