# 移动端开发

> 相比于 `pc` 端开发，移动端开发需要注意的细节较多，比如视口啊，自适应等等，咱们先来看看有哪些需要注意的点

## viewport(视口)

> PC端开发时，咱们基本不需要考虑视口这个属性，但是在移动端开发时不考虑可不行哦，咱们来试试看，给或不给有什么差别

**mdn-Viewport是什么**

**mdn-meta属性**

**viewport深入理解**

咱们先上概念:什么是 `viewport` ，可以叫做视口，是当前文档的可见区域，**大部分时候**和浏览器的窗口大小一致，并且尺寸可变

**无视口代码:**

1. 准备一个基本页面
2. 设置一些基础内容
3. 在浏览器显示之后，切换到移动端界面查看

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Repudiandae a unde nisi amet maxime distinctio voluptatem facilis accusamus totam aspernatur delectus expedita optio ipsum quia, odit fugit sed earum in.</p>
</body>
</html>
```

**显示效果:**

1. 移动端看到的文字十分的小
2. 审查元素可以发现宽度为980(也许不是980，但是肯定比移动端屏幕尺寸大)

**分析:**

1. 不设置视口属性在移动端时为了避免滚动条会把页面的宽度缩放到和屏幕尺寸一致
2. 页面整体缩小了，直接导致显示的文字很小

**添加视口代码:**

1. 在head标签中 `viewport` 相关代码
2. 记得刷新浏览器

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Repudiandae a unde nisi amet maxime distinctio voluptatem facilis accusamus totam aspernatur delectus expedita optio ipsum quia, odit
      fugit sed earum in.
    </p>
  </body>
</html>
```

**显示效果:**

1. 字体变大了
2. 审查元素发现内容的宽度变的和屏幕一致

这就是设置 `viewport` 的作用，现在我们设置的属性是让视口的宽度和设备的宽度一致，缩放值为1，刚好1比1

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

`viewport` 支持设置的属性还有不少，但是现在绝大多数情况下我们设置为上述值即可，这里附上所有属性方便大伙查阅

| Value | 可能值 | 描述 |
| --- | --- | --- |
| `width` | 一个正整数或者字符串 `device-width` | 以pixels（像素）为单位， 定义viewport（视口）的宽度。 |
| `height` | 一个正整数或者字符串 `device-height` | 以pixels（像素）为单位， 定义viewport（视口）的高度。 |
| `initial-scale` | `一个0.0` 到 `10.0之间的正数` | 定义设备宽度（纵向模式下的设备宽度或横向模式下的设备高度）与视口大小之间的缩放比率。 |
| `maximum-scale` | `一个0.0` 到 `10.0之间的正数` | 定义缩放的最大值；它必须大于或等于 `minimum-scale` 的值，不然会导致不确定的行为发生。 |
| `minimum-scale` | 一个 `0.0` 到 `10.0` 之间的正数 | 定义缩放的最小值；它必须小于或等于 `maximum-scale` 的值，不然会导致不确定的行为发生。 |
| `user-scalable` | 一个布尔值（ `yes` 或者 `no` ） | 如果设置为 `no` ，用户将不能放大或缩小网页。默认值为 `yes` 。 |

**小结:**

1. 视口属性设置的是内容的尺寸
2. 移动端不设置视口属性一般内容会比屏幕大，通过压缩实现无横向滚动条，显示效果较差
3. 现在直接把视口属性中的宽度值设置的和设备的宽度一致即可

# rem - 相对单位

> pc端页面开发基本上 `px` 作为单位即可，移动端为了适配屏幕很多时候我们通过 `rem` 会更加便捷，还记得这个单位吗?

mdn-rem

不同于 `px` 绝对单位，写了什么就是什么。 `rem` 是相对单位，相对的是 `html` 标签的 `font-size` ,口说无凭咱们来试试

**测试代码:**

1. 页面中有2个盒子
2. `box` 使用的是 `px` 作为单位
3. `rem-box` 使用的是 `rem` 作为单位
4. `html` 的 `font-size` 为 `10px`
5. 在浏览器中查看显示效果

```
<!DOCTYPE html>
```

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        html{
            font-size: 10px;
        }
        .box{
            width: 100px;
            height: 100px;
            background-color: orange;
        }
        .rem-box{
            width: 10rem;
            height: 10rem;
            background-color: hotpink;
        }

    </style>
    <title>Document</title>
</head>
<body>
    <div class='box'></div>
    <div class='rem-box'></div>
</body>
</html>
```

**显示效果：**

1. 页面中两个盒子
2. 两个盒子的尺寸完全一致都是 `100px*100px`
3. 现在的情况下 `1rem=10px` 刚好就是现在 `html` 元素的字体大小

**调整代码:**

1. 调整 `html` 的字体大小为 `16px`
2. 刷新浏览器查看效果

```html
html{
    font-size: 16px;
}
```

**显示效果:**

1. `rem-box` 的尺寸变为 `160px`
2. 现在 `1rem=16px` 和上面的公式是吻合的
3. 大伙可以随意调整一下 `html` 的字体大小进行测试

虽然 `rem` 用起来并不复杂，但是相比于绝对单位 `px` ,还需要稍微进行计算才可以得到自己要的尺寸，用 `px` 不香吗?

咱们来想象一个场景：

1. 有了一张 `375` 宽度的移动端设计图
2. 通过 `px` 实现了，大概有 `200` 个元素吧
3. 来了一个新的需求， `360` 的屏幕要等比例缩放
4. 重写了一份样式，搞定了
5. 移动端所有的手机都要等比例缩放。。。。
6. 卒。。。。

如果换成 `rem` 呢

1. 有一张 `375` 宽度的移动端设计图
2. 通过某些计算方法把设计图中的尺寸转化为 `rem`
3. 所有的元素都通过 `rem` 写好了
4. 兼容 `360` 屏幕，微微一笑调整一下 `rem` 的大小即可
5. 兼容移动端所有的手机，微微一笑，写段js动态的设置 `rem` 的大小即可

怎么样 听起来是不是挺美的啊，为了让大伙感受的更加直观，咱们来试试看。

# rem - 设计稿尺寸转换

> 重要!!!! 移动端屏幕尺寸各异，但是设计稿一般尺寸只会提供给我们一种，不管给我们的尺寸是多少，先转为 `rem` 先

做 `h5（移动web）` 一般是 `750px` 或更高,如果是 `app` 开发，很大可能是 `375px` ，尺寸无所谓，先看如何转换

**具体步骤：**

1. 假定设计稿的宽度为 `375px`
2. 将设计稿的宽度分为若干份，比如10，20，30.....
3. 上一步的数字无所谓，挑一个好算的即可，这里我们选用 `10`
4. 做个除法 `375/10=37.5` ,现在 `1rem=37.5px`
5. 量取设计稿中的尺寸， `尺寸/37.5` 即可转换为对应的 `rem` 值
6. 现在准备了一份简单的设计稿如下


**试一下:**

1. 分为10份，每一份是 `375/10=37.5px`
2. `1rem=37.5px`
3. 搜索栏高度为: `50/37.5=1.33333333rem` ,除不尽没关系的
4. 轮播图高度为： `150/37.5=4rem`
5. 写个页面，这里只是简单地颜色，通过色块来表示区域

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      body {
        margin: 0;
      }
      html {
        font-size: 37.5px;
      }
      .search {
        height: 1.333333333rem;
        background-color: orange;
      }
      .carousel {
        height: 4rem;
        background-color: skyblue;
      }
    </style>
  </head>
  <body>
    <!-- 搜索区域 -->
    <div class="search"></div>
```

```
    <!-- 走马灯 -->
    <div class="carousel"></div>
  </body>
</html>
```

**显示结果:**

1. 浏览器切换到移动端接口，在屏幕宽度为 `375` 的设备上
2. 搜索区域高度为:50px
3. 轮播图区域高度为:150px
4. 注意：小数点后面的个数别省了哦

我们通过 `rem` 还原了 `375` 尺寸的设计稿，并且在 `375` 尺寸的屏幕上也能够正常显示了，如果切换了屏幕是否会等比例缩放呢?

肯定不会啦，屏幕切换时我们还需要写点东西哦。

## rem - 适配屏幕

> 屏幕宽度改变时，如果不调整 `html` 的字体大小，那么 `rem` 是不会改变的，那么如何调整呢?

还记得上一步中我们将设计图分为了若干份吗，如果你和我一样，那么这个数字应该是 `10` ,如果你换了数字，这里用对应的值即可。

这一次我们来适配一下宽度为 `320` 的屏幕，只要这个想明白了，其他的屏幕做法完全一致

**实现步骤:**

1. 根据屏幕调整 `rem` 大小即可
2. 现在的屏幕宽度为 `320` , `320/10=36`
3. 那么 `1rem=32px` ，对应的调整 `html` 的字体大小即可
4. 屏幕切换到 `320` 的宽度，用 `iphone5/SE` 即可

```
html {
  /* 375屏幕 */
  /* font-size: 37.5px; */
  /* 320屏幕 */
  font-size: 32px;
}
```

**显示效果:**

1. 搜索区域高度: `42.66px`
2. 轮播图区域: `128px`

是不是等比例缩放呢?我们来算算:

- 屏幕宽度比:360/375=0.853333333333333
- 搜索区域比:42.66/50=0.8532
- 轮播图区域比:128/150=0.853333333333333

怎么样，是不是等比例缩放的呀！但是现在并没有自动切换呢，别着急下一步我们就来实现自动切换。

# rem - 自适应

> 通过修改 `rem` 的大小，我们实现了在不同的屏幕上等比例缩放元素，但是现在 `rem` 的设置并不是动态的，而是写死的，如何动态设置呢?

**mdn-视口概念**

在保证设置了视口属性之后，再使用如下代码即可

**实现思路:**

1. `375` 的设计稿，分为了若干份，这里我们使用的是 `10`
2. `375` 的屏幕下: `1rem=37.5px` ，除以10即可
3. 适配 `360` 的屏幕，设置 `1rem=36px` ，也是除以10
4. 当我们设置了 `<meta name="viewport" content="width=device-width, initial-scale=1.0" />` 之后，视口的宽度和屏幕是一致的
5. 那么只需要通过代码获取 `视口宽度/10` 并且把这个值设置给 `html` 的字体大小就可以了嘛

**上代码:**

1. 获取 `html` 元素，并设置 `fontSize`
2. 值是视口大小/10，别忘记拼接 `px` 了哦
3. `document.documentElement.clientWidth` 获取的是视口宽度

```
<script>
    // 设置默认的值

 document.querySelector('html').style.fontSize=document.documentElement.clientWidth/10
+'px'
</script>
```

切换屏幕的时候，我们只需要刷新一下页面就可以了哦。

什么，要在屏幕尺寸改变的时候自动设置啊，通过 `resize` 就能够实现了

**上代码:**

```
  <script>
    // 设置默认的值
    document.querySelector('html').style.fontSize =
document.documentElement.clientWidth / 10 + 'px'
    // 页面尺寸改变时重新计算
    window.onresize = function () {
      document.querySelector('html').style.fontSize =
document.documentElement.clientWidth / 10 + 'px'
    }
  </script>
```

**小结:**

- 通过 `js` 获取视口尺寸并计算出具体的值设置给 `html` 标签即可实现动态设置
- 通过 `resize` 事件即可实现实现页面尺寸再次设置

## rem-flexible

> 自动设置 `rem` 除了可以自己手写以外，也可以使用淘宝推出的 `flexible` 来自动设置，咱们来试试看

**使用Flexible实现手淘H5页面的终端适配**

**github主页**

**使用步骤:**

1. 保证设置了视口属性的前提下: `<meta name="viewport" content="width=device-width, initial-scale=1.0" />`
2. 下包
3. 导包
4. 切换移动设备， `html` 标签上的字体大小是不是也自动切换了啊

**源码:**

```
(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
  // 设置 1rem 等于视口宽度的 /10
  function setRemUnit () {
    var rem = docEl.clientWidth / 10
    docEl.style.fontSize = rem + 'px'
  }

  // 默认设置唇齿
  setRemUnit()

  // reset rem unit on page resize
  // 当 页面 resize触发时，重新计算rem
  window.addEventListener('resize', setRemUnit)
  window.addEventListener('pageshow', function (e) {
    if (e.persisted) {
      setRemUnit()
    }
  })

  // detect 0.5px supports
  if (dpr >= 2) {
    var fakeBody = document.createElement('body')
    var testElement = document.createElement('div')
    testElement.style.border = '.5px solid transparent'
    fakeBody.appendChild(testElement)
    docEl.appendChild(fakeBody)
    if (testElement.offsetHeight === 1) {
      docEl.classList.add('hairlines')
    }
    docEl.removeChild(fakeBody)
  }
}(window, document))
```

**小结:**

1. 在保证设置了视口属性之后，导入阿里的 `flexible`
2. 也可以实现动态的设置 `rem`

## normalize.css

> 很多现代化浏览器会为元素标签添加一些独有的默认样式，这会带来一定的显示差异，我们可以自己写 `css` 去覆盖，但是有个写好的样式库可以直接使用，他就是 `normalize.css`

**github链接**

**看看官网说明:Normalize.css使浏览器更一致地渲染所有元素，并符合现代标准。它只针对需要 `规范化` 的样式。**

**使用方式:**

1. 下包
2. 导包
3. 因为是 `css` ，只需要保证导入即可，不需要调用什么

**注意:**

1. 因为他只处理需要 `规范化` 的样式，所以和我们平时写的 `初始化样式` 可能略有差异
2. 他的作用通俗来说：**尽可能的抹平不同浏览器的显示差异**

# 移动端组件库选择

> 通过组件库我们可以快速实现某些功能，而不用什么都自己从头实现，如何选择呢

**和vue相关的酷炫的库列表**

**vue移动端相关**

这里截取了 `awesome-vue` 中的移动端部分

*UI frameworks for mobile* (ui库,偏ui，功能较少)

- **Framework7-Vue** - Build full featured iOS & Android apps using Framework7 & Vue.
- **vux** - [Chinese] Vue UI Components based on WeUI.
- **vue-onsenui** - Mobile app development framework and SDK using HTML5 and JavaScript. Create beautiful and performant cross-platform mobile apps. Based on Web Components, and provides bindings for Angular 1, 2, React and Vue.js.
- **Weex** - Weex provides the ability to publish cross platform, so web, Android, and IOS apps can use the same API development functions.
- **weex-eros** - [Chinese] Eros is a app solution based on Weex and Vue, which enables you to use API of Vue, simple and quick development of small and medium app.

*Set of components for mobile* (组件库,ui和功能都有)

- **mint-ui** - Mobile UI elements for Vue.js.
- **vant** - A Vue.js 2.0 Mobile UI From YouZan.

- **cube-ui** - A fantastic mobile ui lib implement by Vue.js 2.
- **mand-mobile** - A mobile UI toolkit, based on Vue.js 2, designed for financial scenes.
- **NutUI** - A Vue.js 2.0 UI Toolkit for Mobile Web

选择不少哦，基于什么来选择呢?

1. 星星多的：说明得到了大伙的肯定
2. 文档完善的：想要的功能基本都能在文档中找到，而不需要去翻源码
3. 功能多的：不然项目开发了一半，临时更换或者自己写一个都得不偿失
4. 。。。每个人的选择依据不同，上述选项仅供参考

本次项目中选用 `vant` 因为他:

1. `stars` 数挺高
2. **更新** 频繁，有bug及时修复
3. 功能也基本上都涵盖了

# vant简介

> 引用自官网： 轻量、可靠的移动端 Vue 组件库

Vant 是有赞商城前端开发团队开发的一个基于 Vue.js 的移动端组件库，它提供了非常丰富的移动端功能组件，简单易用。

- **官方文档**
- **GitHub 仓库**

**特性**

- 60+ 高质量组件
- 90% 单元测试覆盖率
- 完善的中英文文档和示例
- 支持按需引入
- 支持主题定制
- 支持国际化
- 支持 TS
- 支持 SSR

**浏览器支持**

现代浏览器以及 Android 4.0+, iOS 8.0+。

# vant基本使用

> 基本上基于 `vue` 的组件库，无论 `pc` 和 `移动` ，用法都是大同小异的

vant-快速上手

**基础项目创建:**

在终端中输入如下命令即可查看版本

```
vue --version
```

如果上述命令没有任何反应，要先行安装哦

```
npm install --global @vue/cli
```

确保安装完毕之后咱们来创建项目，名字要起得有意义

```
vue create 项目名
```

- default：默认选型，基本使用够啦

我们这里直接选择 `default` 就好了

稍等片刻等待项目创建完毕之后,进入项目文件夹，并运行项目

```
cd 项目文件夹
npm run serve
```

**安装vant:**

注意：确保在项目的文件夹下运行如下命令

```
# 通过 npm 安装
npm i vant -S

# or 通过 yarn 安装
yarn add vant
```

**导入vant**

我们这里使用全部导入的方式,在 `main.js` 中的

```
import Vue from 'vue'
import App from './App.vue'
import Vant from 'vant'
import 'vant/lib/index.css'

Vue.use(Vant)

Vue.config.productionTip = false

new Vue({
  render: h => h(App)
}).$mount('#app')
```

随便找个组件测试一下，比如 **按钮**

在 `app.vue` 中整合如下代码

```
<template>
  <div id="app">
    <van-button type="default">默认按钮</van-button>
    <van-button type="primary">主要按钮</van-button>
    <van-button type="info">信息按钮</van-button>
    <van-button type="warning">警告按钮</van-button>
    <van-button type="danger">危险按钮</van-button>
  </div>
</template>
```

如果上述整合没有问题，那么应该在浏览器中就可以看到对应的按钮了，oh-yeah!

# vant使用进阶

> Vant 中的样式默认使用 `px` 作为单位，为了使用 `rem` ，我们还需要做些设置哦

- **vue-cli中postcss**
- **postcss-pxtorem** 是一款 postcss 插件，用于将单位转化为 rem
- **lib-flexible** 用于设置 rem 基准值

**整合 `postcss-pxtorem` :**

- 安装模块

```
npm install postcss-pxtorem --save-dev
```

- 项目根目录创建 **`postcss.config.js`**
- 整合如下代码

```
module.exports = {
    plugins: {
      'postcss-pxtorem': {
        // 转化的基准值 这里使用的是 375/10
        rootValue: 37.5,
        // 哪些元素需要进行转换，*指的是所有
        propList: ['*'],
      },
    },
  };
```

**重新运行项目:**

- 审查元素查看 `vant` 的组件中的样式中的尺寸是不是已经变成了 `rem`

### 整合 `flexible`

- 下包

```
# npm 下载安装
npm i amfe-flexible
```

- main.js中导入

```
import Vue from 'vue'
import App from './App.vue'
// 导入flexible 动态设置rem大小
import 'amfe-flexible'
// 整合vant
import Vant from 'vant'
import 'vant/lib/index.css'

Vue.use(Vant)

Vue.config.productionTip = false

new Vue({
  render: h => h(App)
}).$mount('#app')
```

刷新浏览器， `html` 标签是不是已经设置了字体大小啦！

# vue-cli中的css预处理器

> css预处理器可以让我们用处理器 `独有的` 语法来编写css，更加简单，快捷以及强大，如何在脚手架中使用呢?

vue-cli#预处理器

**安装:**

可以在创建项目时选择需要的处理器，咱们之前选择的好像都是 `default` ，那么可以使用第二种方式来安装

- 在项目根目录
- 根据需求选择安装对应的模块即可

```
# Sass
npm install -D sass-loader sass

# Less
npm install -D less-loader less

# Stylus
npm install -D stylus-loader stylus
```

**使用:**

- 确保上一步安装完成
- 增加了解析 `导入` 预处理器功能
- 也可以在 `.vue` 文件中通过如下代码编写
- 设置对应的 `lang` 即可

```
<style lang="less">
// less代码
</style>

<style lang="scss">
// sass代码
</style>

<style lang="stylus">
// stylus代码
</style>
```

怎么样，现在是不是已经可以再 `vue-cli` 的项目中使用 `预处理器` 啦，写起来爽歪歪

现在有这么一个需求，我需要将项目中用到的颜色全部定义到一个文件中，组件的样式中直接使用这些变量，如何实现呢？咱们下一节就来看

# vue-cli中的自动导入

> 实际项目中 样式的编写 一般会抽取一些公共的颜色，变量...如何实现全局共享呢？

vue-cli#自动化导入

vue-cli-plugin-style-resources-loader

path.resolve

首先咱们来创建一个保存变量的 `less` 文件

就创建在 `/src/style/variable.less` 这里吧

现在在任何一个组件中使用变量 `@bgc` 都是访问不到的，别着急

```
// 定义颜色变量 bgc
@bgc: #0094ff;
```

安装 `vue-cli-plugin-style-resources-loader`

```
vue add style-resources-loader
```

**你没准会看到这个提示：**

```
# 警告，仓库中有没有 commit的更改，最好先处理他们
 WARN  There are uncommited changes in the current repository, it's recommended to commit or stash them first.
# 是否继续
? Still proceed? (y/N)
```

咱们不管他，选择 `y` ，回车并继续

稍等片刻应该看到：

```
# 插件安装完毕
✔  Successfully installed plugin: vue-cli-plugin-style-resources-loader

# 选择使用的 css预处理器
? CSS Pre-processor?
  SCSS
  SASS
  Stylus
❯ Less
```

大伙可以根据需求选择，我们这里选择 `less`

稍等片刻

```
🚀  Invoking generator for vue-cli-plugin-style-resources-loader...
⚓  Running completion hooks...

# 调用插件生成器成功
✔  Successfully invoked generator for plugin: vue-cli-plugin-style-resources-loader
 DONE  style-resources-loader
 # 还需要异步，在 vue.config.js 中添加需要自动导入的资源地址
 One more step, add patterns for your resources's files in vue.config.js
```

在项目的根目录中会自动帮我们创建 `vue.config.js`

```
module.exports = {
  pluginOptions: {
    'style-resources-loader': {
      preProcessor: 'less',
      patterns: []
    }
  }
}
```

还差最后一步啦，我们来配置一下解析的文件,这里直接参照插件的示例代码

- 默认添加 `variable.less` 的解析
- 如果要解析多个地址，`patterns` 是一个数组中，用逗号隔开即可
- 如果要解析某个文件夹下的所有 `.less` 文件呢?**传送门**

```
const path = require('path')
module.exports = {
  pluginOptions: {
    'style-resources-loader': {
      preProcessor: 'less',
      patterns: [
        // 自动导入 variable.less
        path.resolve(__dirname, './src/style/variable.less'),
        // 自动导入 /style 文件夹下的所有.less文件 如果有多个文件这么写也可以
        // path.resolve(__dirname,'./src/style/*.less')
      ]
    }
  }
}
```

搞定，经过上述的配置，我们现在就可以使用定义在 `variable.less` 中的变量了。

比如 `app.vue` 的 `style` 中，或者创建几个新的组件试试

# 小结

> 移动端项目开发中所需的前置知识咱们已经演示完毕了，细节不少，配置也不少，但是。

注意：

1. 项目开发中基础的配置在最开始搭建完毕之后，后面直接使用即可
2. 这部分内容一般是前端的架构负责搭建，但是你最起码得知道什么哦
3. 自己动手试试看，并没有你想象的复杂哦