Css 部分：

移动端布局适配方案：

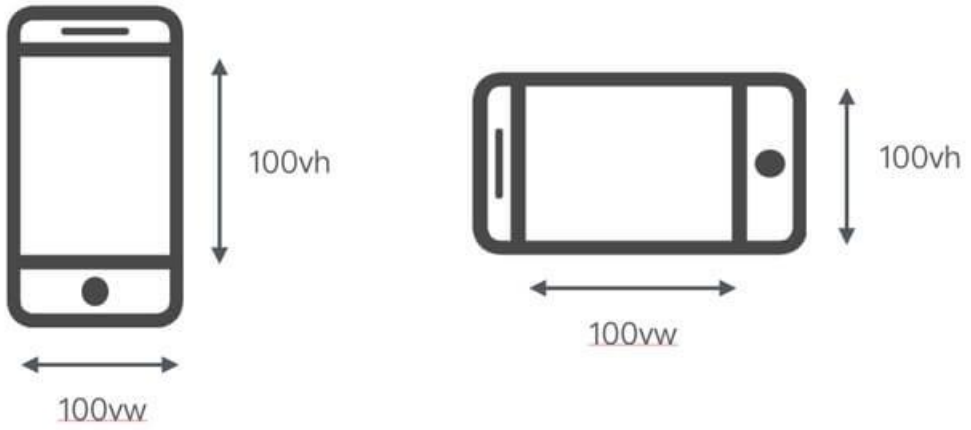
1.百分比和 em/rem 布局

注：该方法宽度可用百分比布局，但高度不适用百分比布局，高度采用固定像素，px/rem/em

2. vw,vh 视口 适配页面

pc 端：浏览器的可视区域，

移动端：1vw 为视口宽度的 1%，lvh 为视口高度的 1%，若手机横屏，相应的 vw 和 vh 也相应的改变



1px 物理像素（普通屏幕下 1px，高清屏下 0.5px），使用 transform 的 scale 实现

```
.divs {
    position: relative;
    &::after {
        content: " ";
       position: absolute;
       left: 0;
       bottom: 0;
       right: 0;
       height: 1px;
       border-bottom: 1px solid #C7C7C7;
       color: #C7C7C7;
       -webkit-transform-origin: 0 100%;
       transform-origin: 0 100%;
       -webkit-transform: scaleY(0.5);
       transform: scaleY(0.5);

    }
```

数组的几种常用方法

1. forEach() : 遍历数组，不会返回新数组

2. map(): 同样是遍历数组，但会返回一个新数组

```
var data = [1,2,3,4,5];
var data1 = data.map(function(value){
   return ++ value;
});
```

3. Filter():遍历匹配，返回包含匹配到元素的新数组

```
var data = [1,2,3,4,5];
var data1 = data.filter(function(value){
   return value <= 3;
});
```

4. Eevery();匹配筛选，返回 true/false，当数组中的所有元素都满足条件时：返回 true，否则返回 false

```
var data = [1,2,3,4,5];
var data1 = data.every(function(value){
   return value < 4;
});
```

5. Some():匹配，返回 true/false，和 every 相反，当数组中有一个元素满足条件就返回 true，全部不满足返回 false，

```
var data2 = data.some(function(value){
 return value >4;
});
```

6. Reduce()使用指定的函数将数组元素进行组合，生成单个值。

```
var sum = data.reduce(function(a,b){
 return a+b;
}); // 若第二个参数没有值，初始值为数组的第一个元素，1+2，3+3，6+4，
   //10+5 =15
var sum1 = data.reduce(function(a,b){
   return a+b;
},5)
   //第二个参数有值，初始值为第二个参数，5+1，6+2，8+3，11+4，15+5=20
```

7. reduceRight()：和 reduce 类似，不同：按照数组索引从高到低处理数组，而不是正常的从高到低

```
var data = ['a','b','c'];
var str = data.reduce(function(x,y){ //顺序
 return x+y;
});  //’abc’
```

```
var str1 = data.reduceRight(function(x,y){ //逆序
 return x+y;
}); //’cba’
```

8. indexOf(): 从头至尾搜索数组中具有给定值的元素，找到第一个就返回元素的索引，不再往下找，找不到返回-1

```
var data = ['a','b','a','c','a'];

console.log(data.indexOf('a')); //0
console.log(data.indexOf('d')); //-1
console.log(data.lastIndexOf('a'));//4

console.log(data.lastIndexOf('a',-2));//2 从倒数第二个开始
console.log(data.lastIndexOf('a',1));//0  从顺序第二个往前
```

9.数组的深拷贝

```
this.list = [...res.lists].map(e=>{
      return {
        ...e,
         type:’input’
      }
 })
```

下边的 input 会覆盖上边的 e 里边的值

```
this.list = [...res.lists].map(e=>{
     return Object.assign(e,{
         type: ’ input ‘,
     })
})
```

Object.assign:后边的会覆盖前边的对象里边的内容，注：使用 map 时最好 return 一个新列表出来，这样不会改动原来的东西