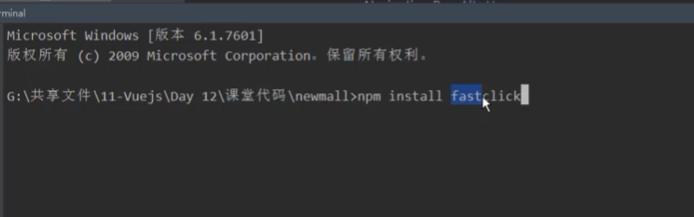
Better-scroll使用实例（分类组件导出）

<template>  
 <div class="wrapper">  
 <!--1.无论是否设置click:false, button都可以点击-->  
 <button @click="btnClick">按钮</button>  
  
 <!--2.必须设置click: true, 那么div才能监听点击-->  
 <div @click="divClick">哈哈呵呵</div>  
 <ul class="content">  
 <button @click="btnClick">按钮</button>  
 <li>分类列表1</li>  
 <li>分类列表2</li>  
 <li>分类列表3</li>  
 <li>分类列表4</li>  
 <li>分类列表5</li>  
 <li>分类列表6</li>  
 <li>分类列表7</li>  
 <li>分类列表8</li>  
 <li>分类列表9</li>  
 <li>分类列表10</li>  
 <li>分类列表11</li>  
 <li>分类列表12</li>  
 <li>分类列表13</li>  
 <li>分类列表14</li>  
 <li>分类列表15</li>  
 <li>分类列表16</li>  
 <li>分类列表17</li>  
 <li>分类列表18</li>  
 <li>分类列表19</li>  
 <li>分类列表20</li>  
 <li>分类列表21</li>  
 <li>分类列表22</li>  
 <li>分类列表23</li>  
 <li>分类列表24</li>  
 <li>分类列表25</li>  
 <li>分类列表26</li>  
 <li>分类列表27</li>  
 <li>分类列表28</li>  
 <li>分类列表29</li>  
 <li>分类列表30</li>  
 <li>分类列表31</li>  
 <li>分类列表32</li>  
 <li>分类列表33</li>  
 <li>分类列表34</li>  
 <li>分类列表35</li>  
 <li>分类列表36</li>  
 <li>分类列表37</li>  
 <li>分类列表38</li>  
 <li>分类列表39</li>  
 <li>分类列表40</li>  
 <li>分类列表41</li>  
 <li>分类列表42</li>  
 <li>分类列表43</li>  
 <li>分类列表44</li>  
 <li>分类列表45</li>  
 <li>分类列表46</li>  
 <li>分类列表47</li>  
 <li>分类列表48</li>  
 <li>分类列表49</li>  
 <li>分类列表50</li>  
 </ul>  
 </div>  
  
</template>  
  
<script>  
 import BScroll from 'better-scroll'  
 export default {  
 name: "Category",  
 //created为组件创建完成后调用的钩子，但页面上还没渲染dom  
 /\* created() {  
 new BScroll('.wrapper', {  
 })  
 }\*/  
 data() {  
 return {  
 scroll: null  
 }  
 },  
  
 /\*要用better-scroll，滑动的内容（content）的外层只能包裹一个标签（wrapper）\*/  
 // console.log(BScroll);  
 //probe侦测(值为0和1都是不侦测实时的位置，值为2在手指滚动的过程中侦测，手指离开后的惯性滚动过程中不侦测)  
 //值为3：只要是滚动，都侦测（包括惯性)  
 mounted() {  
 //这里传入需要滚动的区域content  
 const bscroll = new BScroll('.wrapper', {  
 click: true,  
 probeType: 3,  
 pullUpLoad: true  
 })  
 bscroll.on('scroll', (position) => {  
 ***console***.log(position);  
 })  
 bscroll.on('pullingUp', () => {  
 ***console***.log('上拉加载更多');  
 //发送网络请求，请求更多数据  
  
 //等数据请求完成，并且将新的数据展示出来后, bscroll.finishPullUp()进行下一次的上拉加载  
 setTimeout(() => {  
 bscroll.finishPullUp()  
 },2000)  
  
 })  
 },  
 methods: {  
 btnClick() {  
 ***console***.log('aksdkak');  
 },  
 divClick() {  
 ***console***.log('aksdkak');  
 }  
 }  
 }  
</script>  
  
<style scoped>  
 .wrapper {  
 height: 150px;  
 background-color: #f00;  
  
 /\*原生的局部滚动\*/  
 /\*overflow-y: scroll;\*/  
  
 /\*使用了better-scroll\*/  
 overflow: hidden;  
 }  
</style>



解决移动端300ms延迟



二、图片懒加载（vue-lazyload）

图片需要显示在屏幕上时，再加载

1. npm install vue-LazyLoad --save
2. 

//3.使用懒加载的插件 //4.修改img的src :src="showImg" -> v-lazy="showImg"  
Vue.use(***VueLazyLoad***, {  
 loading: require('./assets/img/common/lazy.png') //导入占位图  
})

1. 修改像素单位为适配单位（使用插件来完成，在打包时将所有px单位转为vw）

module.exports = {  
 plugins: {  
 autoprefixer: {},  
 "postcss-px-to-viewport": {  
 //iPhone6的设计稿  
 viewportWidth: 375, //视口的宽度  
 viewportHeight: 667, //视口的高度  
  
 unitPrecision: 5, //保留5位小数，指定‘px‘转换为视窗单位值的小数位数（很多时候无法整除）  
 viewportUnit: 'vw', //指定需要转换成的视窗单位，建议使用vw  
 selectorBlackList: ['ignore', 'tab-bar'], //指定不需要转换的类，  
  
 minPixelValue: 1, //小于等于1px的不作转换为视窗单位  
 mediaQuery: false, //允许在媒体查询中转换’px‘  
 exclude: [/TabBar/] //只能使用正则排除TabBar组件不被转换  
 }  
 }  
}

1. 补充：也可以用px2rem将px单位转为rem单位
2. 服务器部署



1. xshell连接到阿里云服务器，

yum intsall nginx

systemctl start nginx.service //开启nginx服务

systemctl enable nginx.service //跟随系统启动

cd ..

cd etc/

cd nginx/

vim nginx.conf

或者直接使用FileZilla软件直接打开nginx.conf