**目录**

[路由以及出传值 2](#_Toc59879201)

[获取参数 3](#_Toc59879202)

[Get请求 3](#_Toc59879203)

[js把时间戳转为为普通日期格式 3](#_Toc59879204)

[js把普通日期格式转为时间戳 3](#_Toc59879205)

[正则表达式 4](#_Toc59879206)

[prototype含义 4](#_Toc59879207)

[VUEX的使用 4](#_Toc59879208)

[VUEX的方法调用 5](#_Toc59879209)

[将数据放入浏览器缓存中和取出方法 6](#_Toc59879210)

[JS数组操作 6](#_Toc59879211)

[使用 splice() 截取数组 7](#_Toc59879212)

[使用 slice() 截取数组 8](#_Toc59879213)

[CSS中的弹性布局 9](#_Toc59879214)

[VUE 子组件向父组件传值 9](#_Toc59879215)

[VUE 子组件向父组件传递方法 10](#_Toc59879216)

[CSS动画 12](#_Toc59879218)

[CSS3动画（transition，transform）和元素的隐藏显示display 12](#_Toc59879219)

[CSS动画：transition 的用法介绍 13](#_Toc59879220)

[等待dom渲染完成之后执行函数 14](#_Toc59879221)

[vue路由传参的三种基本方式 14](#_Toc59879222)

[富文本编辑器 15](#_Toc59879223)

[JS解析JSON数据 17](#_Toc59879224)

[JS获取子元素 17](#_Toc59879225)

[JavaScript之三元运算符 17](#_Toc59879226)

[JavaScrip字符串转数字 17](#_Toc59879227)

[Vue监听器 17](#_Toc59879228)

[JavaScrip将小写数字转化成大写 18](#_Toc59879229)

[JavaScrip修改this指向 18](#_Toc59879230)

[原生JS添加样式 18](#_Toc59879231)

[setAttribute设置属性 19](#_Toc59879232)

[setAttribute设置事件 19](#_Toc59879233)

[原生JS去除同级样式 19](#_Toc59879234)

[原生JS去隐藏滚动条 19](#_Toc59879235)

[原生JS绘图 20](#_Toc59879236)

[JS  阻止点击事件 20](#_Toc59879237)

[解决vue三级路由报错 20](#_Toc59879238)

[Vue中bus的使用 21](#_Toc59879239)

[CSS渐变 21](#_Toc59879240)

[Flex布局 21](#_Toc59879241)

## 路由以及传值

      ReturnToEnterpriseList(){

            this.$router.push({

                path:'./index',

                query:{//携带参数

                    DataHttp:'?ReturnToEnterpriseList',

                }

            });

        }

## 获取参数

    mounted() {

*var* GetDataHttp =this.$route.query.DataHttp;

        console.log(GetDataHttp);

    },

## Get请求

getappUser(){

*var* data = '?page='+ this.page + '&limit='+ this.limit + '&phoneNumber=' + this.input + '&role=' + this.model1

    this.$axios.get('admin/user/all/app/user'+ data).then((*res*) *=>* {

        console.log(res)

    }).catch((*err*)*=>*{

    })

},

## js把时间戳转为为普通日期格式

getLocalTime(*nS*) {

    return new Date(parseInt(nS) \* 1000).toLocaleString().replace(/:\d{1,2}$/,' ');

},

## js把普通日期格式转为时间戳

方法1

        fgetLocalTime() {*//将时间转换成时间撮*

*let* date = new Date(*this*.value2[0]);

*let* start = date.getTime(date);

*this*.start = start;

*let* date1 = new Date(*this*.value2[1]);

*let* end = date1.getTime(date1);

*this*.end = end;

        },

方法2

*var* date = new Date( val.startTime); {*//将时间撮转换成时间*

*var* time1 = date.getFullYear() + '-' + (date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() +1) + '-' + date.getDate();

*var* date1 = new Date(val.endTime);

*var* time2 = date.getFullYear() + '-' + (date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() +1) + '-' + date.getDate();

       DataAjax7[index].col7 = time1+'/'+ time2;

当时间为字符串时

var date = new Date(parseInt(val.createTime));

                                var time1 =

                                    date.getFullYear() +

                                    '-' +

                                    (date.getMonth() + 1 < 10 ? '0' + (date.getMonth() + 1) : date.getMonth() + 1) +

                                    '-' +

                                    date.getDate();

## 正则表达式

// 使用 test()

test() //方法是一个正则表达式方法。

test() //方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

//以下实例用于搜索字符串中的字符 "e"：

//实例

*var* patt = /e/;//定义正则

patt.test("The best things in life are free!"); //使用正则去验证

## prototype含义

*Vue*.prototype.$axios = axios;//给Vue函数添加一个原型属性$axios,指向Axios,这样做的好处是在vue实例或组件中不用再去重复引用Axios,直接用this.$axios就能执行axios 方法了

## VUEX的使用

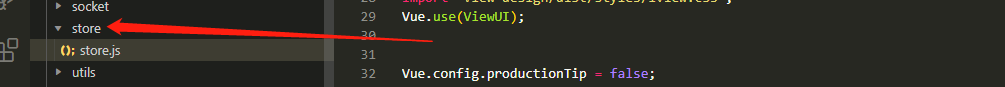
1.安装npm install vuex –save

2.再main.js中引入文件

import Vuex from 'vuex';

Vue.use(Vuex)

3.在src下创建store文件夹,并写入store.js文件



4. store.js文件中写入

import Vue from 'vue';

import Vuex from 'vuex';

Vue.use(Vuex)

*const* store = new Vuex.Store({

    state: {

        todos: [

            { id: 1, text: '...', done: true },

            { id: 2, text: '...', done: false }

        ]

    },

    getters: {

        doneTodos: *state* *=>* {

        return *state*.todos.filter(*todo* *=>* *todo*.done)

        }

    }

})

export default store ;

5. 在main.js文件中导入store文件

import store from './store/store.js';

注意：需要在vue实例上挂载store

new Vue({

    router,

    VSchart,

    store,

    render: *h* *=>* h(App)

}).$mount('#app');

## VUEX的方法调用

1.调用getter 的数据的方法

*const* store = new Vuex.Store({

    state: {

        // tabela--orderlistb

        orderlistbdata:[],

    },

    getters: {

        doneTodos: *state* *=>* {

            // return state.orderlistbdata

        }

    },

})

使用this.$store.getters.方法名

console.log(this.$store.getters.doneTodos)

2. 调用mutations 的数据的方法

*const* store = new Vuex.Store({

    state: {

      count: 1

    },

    mutations: {

        increment (*state*, *n*) {

*state*.count += *n*

        }

      }

  })

使用this.$store.commit(‘方法名’,value)

   this.$store.commit('increment',30)//先提交数据并修改

   console.log(this.$store.state)//调用修改后的数据

注：当提交中携带参数value时，在mutation里必须要传入对的参数，如state02

*const* store = new Vuex.Store({

    state: {

        as : '',

    },

    getters: {

        doneTodos: *state* *=>* {

            // return state.orderlistbdata

        }

    },

    mutations: {

        doneTodosData(*state*,*state02*){

            return *state*.as = *state02*

        }

    },

    actions: {

    },

})

在客户端存储数据：

　　HTML5提供了2种在客户端存储数据的新方法：localStorage没有时间期限，除非将它移除，sessionStorage即会话，当浏览器关闭时会话结束，有时间期限，具有自行百度。

　　之前，这些都是由cookie完成的，但是cookie不适合大量数据的存储，因为它们由每个对服务器的请求来传递，这使得cookie速度很慢，而且效率不高。

web存储分为localStorage个sessionStorage。

## 将数据放入浏览器缓存中和取出方法

            localStorage.setItem('idd',this.$store.state.Vuxorderlistbdata)

            localStorage.getItem('idd')

## JS数组操作

JS删除第一个元素，返回剩下的元素

           this.pageid.shift();

JS删除最后一个元素，并返回被删除的元素

*this*.data6.push(bbb)

JS将数字转换成字符串

*var* D = *this*.DataData;

*var* Datachange = D.toString();

            console.log(typeof Datachange)

JS将字符串转换成数字

*var* dseas = parseInt('DDDDDDDD')

            console.log(dseas )   *// 输出结果NaN*

*//    NaN 指 not a number 不是一个数值~*

*//    NaN 属性是代表非数字值的特殊值。该属性用于指示某个值不是数字*

*//    然而  NaN === NaN  结果为 false，原因是NaN只能指明某个值不是数字类型*

*//    但不是数字类型的东西很多*

JS将字符串分割并转换成数组

*let* transformationArr = DatachangeData.split(',');

            console.log(transformationArr)

注：当字符串没有空格是不会被分割二直接转换成数组

JS截取数组的某个字符或字符串

*var* str='12345';

*var* s=str.substr(0,1);*//substr,第一个参数是开始索引(从0开始)，第二参数是截取的字符串的长度*

JS读取字符串的某一个字符

*var* str="Hello world!";

document.write(str.charAt(0)); *//charAt()方法可返回指定位置的字符。*

## 使用 splice() 截取数组

splice() 方法可以添加元素、删除元素，也可以截取数组片段。删除元素时，将返回被删除的数组片段，因此可以使用 splice() 方法截取数组片段。  
由于 splice() 方法的功能多，参数复杂，使用时应该注意下面几个问题。  
1) splice() 方法的参数是可选的。如果不给它传递参数，则该方法不执行任何操作。如果给它传递一个参数，则该方法仅执行删除操作，参数值指定删除元素的起始下标，（包含该下标元素），splice() 方法将删除后面所有元素。

* 如果指定两个参数，则第 2 个参数值表示要删除元素的个数。
* 如果指定三个或多个参数，则第 3 个以及后面所有参数都被视为插入的元素。
* 如果不执行删除操作，第 2 个参数值应该设置为 0，但是不能够空缺，否则该方法无效。

2) splice() 方法的删除和插入操作是同时进行的，且是在原数组基础上执行操作。插入的元素将填充被删除元素的位置，并根据插入元素个数适当调整插入点位置。而不是在删除数组之后重新计算插入点的位置。  
3) splice() 方法执行的返回值是被删除的子数组。

*var* a = [1,2,3,4,5];  *//定义数组*

*var* b = a.splice(2);  *//从第三个元素开始执行删除*

  console.log(b);  *//被删除的子数组是[1,2]*

如果没有删除元素，则返回的是一个空数组。

*var* b = a.splice(2,0);  *//不执行删除操作*

  console.log(b.constructor == Array);  *//返回true，说明是一个空数组*

4) 当第 1 个参数值大于 length 属性值时，被视为在数组尾部执行操作，因此删除无效，但是可以在尾部插入多个指定元素。

*var* a = [1,2,3,4,5];  *//定义数组*

*var* b = a.splice(6,2,2,3);  *//起始值大于length属性值*

  console.log(a);  *//返回[1,2,3,4,5,2,3]*

5) 参数取负值问题。如果第 1 个参数为负值，则按绝对值从数组右侧开始向左侧定位。如果第 2 个参数为负值，则被视为 0。

*var* a = [1,2,3,4,5];   *//定义数组*

*var* b = a.splice(-2,-2,2,3);  *//第一、二个参数都为负值*

  console.log(a);  *//返回[1,2,3,2,3,4,5]*

## 使用 slice() 截取数组

slice() 方法与 splice() 方法功能相近，但是它仅能够截取数组中指定区段的元素，并返回这个子数组。该方法包含两个参数，分别指定截取子数组的起始和结束位置的下标。

*var* a = [1,2,3,4,5];  *//定义数组*

*var* b = a.slice(2,5);  *//截取第三个元素到第六个元素前的所有元素*

console.log(b);  *//返回[3,4,5]*

使用该方法时，应该注意下面几个问题。  
1) 第 1 个参数指定起始下标位置，包括该值指定的元素；第 2 个参数指定结束位置，不包括指定的元素。  
2) 该方法的参数可以自由设置。如果不传递参数，则不会执行任何操作；如果仅指定一个参数，则表示从该参数值指定的下标位置开始，截取到数组的尾部所有元素。

*var* b = a.slice(2);  *//截取数组中第三个元素，以及后面所有元素*

  console.log(b);   *//返回[3,4,5]*

3) 当参数为负值时，表示按从右到左的顺序进行定位，即倒数定位法，而不再按正数顺序定位（从左到右），但取值顺序依然是从左到右。

*var* b = a.slice(-4,-2);  *//截取倒数第四个元素到倒数第二个元素前的元素*

console.log(b);   *//返回[2,3]*

如果起始下标值大于或等于结束下标值，将不执行任何操作。

*var* b = a.slice(-2,-4);  *//截取倒数第二个元素到倒数第四个元素前的元素*

  console.log(b);  *//返回空集*

上面示例说明数组在截取时，始终是按从左到右的顺序执行操作，而不会是从右到左的反向操作。  
4) 当起始参数值大于或等于 length 属性值时，将不会执行任何操作，返回空数组。而如果第二个参数值大于 length 属性值时，将被视为 length属性值。

*var* b = a.slice(3,10);  *//截取第四个元素，直到后面所有元素*

  console.log(b);  *//返回[4,5]*

5) slice() 方法将返回数组的一部分（子数组），但不会修改原数组。而 splice() 方法是在原数组基础上进行截取。如果希望在原数组基础上进行截取操作，而不是截取为新的数组，这时候就只能够使用 splice() 方法了。

## CSS中的弹性布局

父级元素加上：

display: flex;

子集元素加上

    flex-grow:1;

则水平排列元素的第一个元素五发设置固定宽度

解决方案在子元素加上

    max-width:200px;

    min-width: 200px;

## VUE 子组件向父组件传值

1.子组件定义传值方法

<template>

    <div *class*="app">

       <input @*click*="sendMsg" *type*="button" *value*="给父组件传递值">

    </div>

</template>

<script>

export default {

    data () {

        return {

*//将msg传递给父组件*

            msg: "我是子组件的msg",

        }

    },

     methods:{

         sendMsg(){

*//func: 是父组件指定的传数据绑定的函数，this.msg:子组件给父组件传递的数据*

*this*.$emit('func',*this*.msg)

         }

     }

}

</script>

2.父组件定义接收值方法

<template>

    <div *class*="app">

        <vuexrequestdata @*func*="getMsgFormSon"></vuexrequestdata>

    </div>

</template>

<script>

import vuexrequestdata from '../Vuex-Request-Data-xyr/vuexrequestdata.vue'

export default {

    data () {

        return {

            msgFormSon: "this is msg"

        }

    },

    components:{

        vuexrequestdata,

    },

    methods:{

            getMsgFormSon(*data*){

*this*.msgFormSon = data

                console.log(*this*.msgFormSon)

            }

    }

}

</script>

## VUE 子组件向父组件传递方法

1.子组件定义

<template>

  <div>

    <h1 @*click*="clickAction">小鱼儿</h1>

  </div>

</template>

<script>

  export default {

    methods:{

      clickAction(){

*this*.$emit('clickHeadAction',123);

      }

    }

  }

</script>

2.父组件定义

<template>

  <div>

    <sendeventtofa @*clickHeadAction*='clickChild'></sendeventtofa>

  </div>

</template>

<script>

import sendeventtofa from '../SendEvent-to-Fa-xyr/sendeventtofa';

  export default {

    components:{

      sendeventtofa,

    },

    methods:{

      clickChild(*msg*){

          alert(msg)

      }

    }

  }

</script>

注释：传递方法不用：而用@，：绑定属性@绑定事件

## CSS动画

@keyframes *myfirst*

{

  from {width: 100px;}

  to {width:800px;}

}

@-webkit-keyframes *myfirst* */\* Safari and Chrome \*/*

{

  from {width: 100px;}

  to {width:800px;}

}

*.divdiv*

{

  width:100px;

  height:100px;

  background:red;

  animation:myfirst 5s;

*-webkit-animation*:myfirst 5s; */\* Safari and Chrome \*/*

}

## CSS3动画（transition，transform）和元素的隐藏显示display

**transition（过度动画，从某某某属性到某某某属性）**

transition: 属性持续时间定时功能延迟

|  |  |
| --- | --- |
| **值** | **描述** |
| [transition-property](https://www.w3school.com.cn/cssref/pr_transition-property.asp) | 规定设置过渡效果的 CSS 属性的名称。 |
| [transition-duration](https://www.w3school.com.cn/cssref/pr_transition-duration.asp) | 规定完成过渡效果需要多少秒或毫秒。 |
| [transition-timing-function](https://www.w3school.com.cn/cssref/pr_transition-timing-function.asp) | 规定速度效果的速度曲线。 |
| [transition-delay](https://www.w3school.com.cn/cssref/pr_transition-delay.asp) | 定义过渡效果何时开始。 |

**transform （转化动画）**

   旋转：rotate()

   缩放：scale()

   翻转：skew()

**transition和transform 动画会占文档流**

**display：none   不占文档流    block 占文档流**

## CSS动画：transition 的用法介绍

CSS中最简单的动画叫做 *transition(转变)*。通常，当一个元素的样式属性值发生变化时，我们会立即看到页面元素发生变化，也就是页面元素从旧的属性值立即变成新的属性值的效果。*Transition(转变)*能让页面元素不是立即的、而是慢慢的从一种状态变成另外一种状态，从而表现出一种动画过程。

这些转变效果可以用下列属性来指定：

*transition-property* – 什么属性将用动画表现，例如, opacity。

*transition-duration* – 转变过程持续时间。

*transition-timing-function* – 转变时使用的调速函数(比如, linear、 ease-in 或自定义的 cubic bezier 函数)。

*transition* – 三种属性的合体简写。

下面是一个例子：

div {

opacity: 1;

transition: opacity 1s linear;

}

div:hover {

opacity: 0;

}

## 等待dom渲染完成之后执行函数

        GetDataAjax(){

*var* url = 'admin/company/activity/sys/flash/list';

*var* data = {

                "activityTitleLike":*this*.value01,

                "companyNameLike":*this*.value02,

                "page": *this*.page,

                "status": 1,

                "timeEnd":*this*.value2[1],

                "timeStart":*this*.value2[0],

            }

*this*.$axios.post(url,data).then((*res*)=>{

*var* AjaxData9 = res.data.list[0];

*this*.$nextTick(() => {

*this*.Datar9 = AjaxData9;

                })

            }).catch((*err*)=>{

            })

        }

## vue路由传参的三种基本方式

第一种方法 页面刷新数据不会丢失

methods:{

  insurance(*id*) {

*//直接调用$router.push 实现携带参数的跳转*

*this*.$router.push({

      path: `/particulars/${id}`,

    })

},

需要对应路由配置如下：

    {

     path: '/particulars/:id',

     name: 'particulars',

     component: particulars

    }

可以看出需要在path中添加/:id来对应 $router.push 中path携带的参数。在子组件中可以使用来获取传递的参数值  
另外页面获取参数如下

*this*.$route.params.id

第二种方法 页面刷新数据会丢失

通过路由属性中的name来确定匹配的路由，通过params来传递参数。

  methods:{

  insurance(*id*) {

*this*.$router.push({

      name: 'particulars',

      params: {

        id: id

      }

    })

  },

对应路由配置: 注意这里不能使用:/id来传递参数了，因为组件中，已经使用params来携带参数了。

     {

      path: '/particulars',

      name: 'particulars',

      component: particulars

    }

子组件中: 这样来获取参数

*this*.$route.params.id

        }

## 富文本编辑器

下载

**npm install vue-quill-editor --save  
cnpm install quill --save**

**在页面中引用**

 <div *class*="edit\_container">*<!--附文本-->*

    <quill-editor

*v-model*="content"

*ref*="myQuillEditor"

        :*options*="editorOption"

        @*blur*="onEditorBlur($event)" @*focus*="onEditorFocus($event)"

        @*change*="onEditorChange($event)">

    </quill-editor>

</div>

<script>

    import { quillEditor } from "vue-quill-editor"; *//调用编辑器*

    import 'quill/dist/quill.core.css';

    import 'quill/dist/quill.snow.css';

    import 'quill/dist/quill.bubble.css';

    export default {

        components: {

            quillEditor

        },

        data() {

            return {

                content: `欢迎使用`,

                editorOption: {}

            }

        },

        methods: {

            onEditorReady(*editor*) { *// 准备编辑器*

            },

            onEditorBlur(){}, *// 失去焦点事件*

            onEditorFocus(){}, *// 获得焦点事件*

            onEditorChange(){}, *// 内容改变事件*

        },

        computed: {

            editor() {

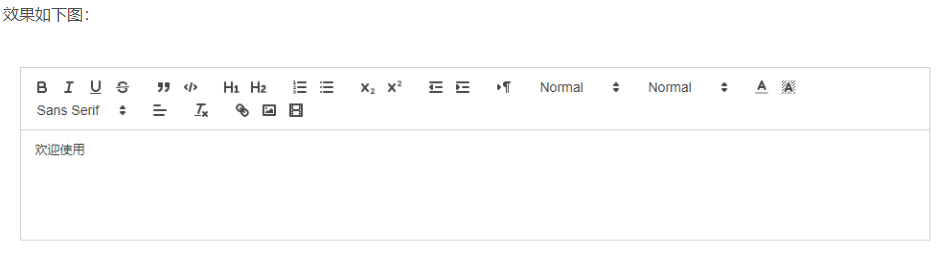
                return *this*.$refs.myQuillEditor.quill;

            },

        }

    }

</script>



        methods: {

        onEditorReady(*editor*) { *// 准备编辑器*

        },

        onEditorBlur(){}, *// 失去焦点事件*

        onEditorFocus(){}, *// 获得焦点事件*

        onEditorChange(){}, *// 内容改变事件*

},

        computed: {

            editor() {

                return *this*.$refs.myQuillEditor.quill;

            },

        },

        components: {

            quillEditor

        },

## JS解析JSON数据

*let* Str = TransVvalue.productDTO.parameter;

*let* Obj = JSON.parse(Str);

console.log(Obj)

## JS获取子元素

*const* childrener = XyeOL[j].children;

## JavaScript之三元运算符

**格式： 表达式1?表达式2:表达式3**

**说明：如果表达式1为true（判断条件） ，则整个表达式的结果就是表达式2的值，如果表达式false，则整个表达式的结果就是表达式3的值.**

*const* Numter = 5 ;

      Numter<3?alert('AAA'):alert('BBB')

## JavaScrip字符串转数字

                            DData.forEach(*function*(*v*,*i*){

                                DData02[i] = Number(v);

                            })

                            console.log(DData02)

## Vue监听器

  data() {

    return {

      message: 'Hello world!',

      watchMsg: '我是监听操作',

    }

  },

  watch:{

*// 需要注意的是这里的监听对象也需要变化*

    'obj.watchMsg': {

      handler(*newVal*,*oldVal*){

        console.log('newVal:'+newVal,'oldVal:'+oldVal);

      },

      immediate:true,*//而immediate:true代表如果在 wacth 里声明了watchMsg之后，就会立即先去执行里面的handler方法，如果为false就跟我们以前的效果一样，不会在绑定的时候就执行。*

      deep:true,*//深度监听，监听对象时需要加上*

    }

  },

## JavaScrip将小写数字转化成大写

        getData(){

*const* prizeLevel = {"1":"一","2":"二","3":"三"};

*this*.detailsData = *this*.$route.query.Data;

*let* strinfer = *this*.$route.query.Data.prizeItemList;

            for(*let* i in strinfer){

                strinfer[i].prizeLevel = prizeLevel[strinfer[i].prizeLevel + ""] + '等奖';

*this*.detailsData.prizeItemList.prizeLevel = strinfer[i].prizeLevel;

            }

        }

## JavaScrip修改this指向

*const* \_this = *this*;

## 原生JS添加样式

<script *type*="text/javascript">

  arr[i].classList.add('active');

  arr[j].classList.remove('active');

  oP.style.background = 'red';

  oP.style.color = 'yellow';

  oP.style.border = '10px #ccc solid';

*var* root = document.getElementsByClassName('root')[0];

  root.style.cssText = 'background-color: blue;color: #fff;';

</script>

setAttribute设置属性

<script>

  elementNode.setAttribute(name,value)*//name属性名，value属性值*

*var* input = document.createElement("input");

  input.setAttribute("type", "text");

  input.setAttribute("name", "q");

  input.setAttribute("class",".className");//清除原来的样式，从新定义样式

</script>

setAttribute设置事件

*var* bar = document.getElementById("uw3c");

  bar.setAttribute("onclick", "javascript:alert('uw3c!');");

用setAttribute设置click属性很好理解，但是这会有兼容性问题，IE中不支持setAttribute设置事件属性，所以为了兼容，只能通过如下方法：

document.getElementById("uw3c").onclick= *function* () { alert("uw3c!"); }

|  |
| --- |
|  |

 XyeOL[i].setAttribute('class','hideol')

## 原生JS去除同级样式

情景1

        changeBtn() {

            for (*let* i = 0; i < *this*.$refs.list.length; i++) {

*this*.$refs.list[i].addEventListener('click', () => {

*this*.$refs.list[i].classList.add('BGcolor');

*//  parentNode  返回当前元素的父节点*

*//  childNodes   获得 body 元素的子节点集合*

*//  nodeType 属性返回被选节点的节点类型*

*var* siblings = *this*.$refs.list[i].parentNode.childNodes;

                    for (*var* j = 0; j < siblings.length; j++) {

                        if (siblings[j].nodeType == 1 && siblings[j] != *this*.$refs.list[i]) {

                            siblings[j].className = '';

                        }

                    }

                });

            }

        }

情景2

<template>

    <div class="titleDtata">

        <ul class="ulData">

            <span class="spanClass">&nbsp;</span>

            <li class="ulliData" v-for="(item, index) in listData" :key="index">

                <div ref="liDivf" class="liDiv">

                    <span ref="listDa" class="titleClass">{{ item.title }}</span>

                    <span ref="ishsow" class="lineClass">/</span>

                </div>

            </li>

        </ul>

    </div>

</template>

<script>

import bus from './bus.js';

export default {

    data() {

        return {

            listData: [

                {

                    title: '产品中心'

                },

                {

                    title: '商家后台'

                },

                {

                    title: '总后台'

                },

                {

                    title: '数据分析'

                },

                {

                    title: '变化便当'

                }

            ]

        };

    },

    mounted() {

        for (let i = 0; i < this.$refs.listDa.length; i++) {

            if (i == this.$refs.listDa.length - 1) {

                //去除最后一个元素的斜线

                this.$refs.liDivf[i].removeChild(this.$refs.ishsow[i]);

            }

            this.$refs.listDa[i].addEventListener('click', () => {

                for (let j = 0; j < this.listData.length; j++) {

                    if (i == j) {

                        this.$refs.listDa[i].classList.add('fontColor');

                    } else {

                        this.$refs.listDa[j].classList.remove('fontColor');

                    }

                }

            });

        }

    },

    methods: {}

};

</script>

## 原生JS去隐藏滚动条

ul::-webkit-scrollbar {

    width: 0 !important;

}

## 原生JS绘图

1. 获取canvas对象的画笔。

　　document.getElementById(" ID名 ").getContext(' 2d ');

2. 设置画笔属性。

            .lineWidth = 5; //设置划线宽度为5像素。

            .strokeStyle = "rgb(250,255,20)";    //设置画线颜色。

            .fillStyle = 'red'; //设置填充颜色。

3. 画矩形

　　.strokeRect(x, y, w, h);     画矩形框

　　.fillRect(x, y, w, h);            画矩形面

4. 路径画图。

　　.beginPath()                 开始路径定义

　　.closePath()                 闭合路径（使首尾相连）

　　.fill()                              填充

　　.stroke()                       划线

　　.moveTo(x, y)              移动画笔到（x,  y）

　　lineTo(x, y)                  从当前位置划线至（x, y）

　　arc(x, y, r, s, e, b)        画弧（xy坐标，r半径， s e开始与结束角度， b 划线方向 true为逆时针）

5. 清除画面内容。

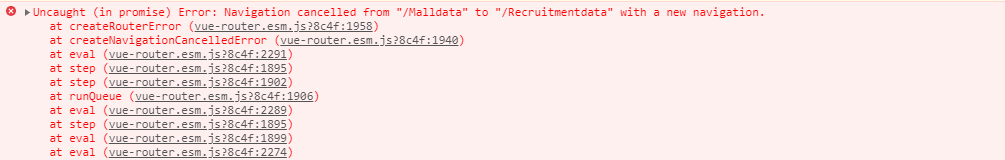
　　.clearRect(x,y,w,h)      擦除矩形中的内容

## JS  阻止点击事件

event.stopPropagation();

事件处理过程中，阻止了事件冒泡，但不会阻击默认行为（它就执行了超链接的跳转）

## 解决vue三级路由报错



注意：三级路由以及多级路由嵌套需要用挂载的方式

解决方案：**在main.js中写入以下代码**

import Router from 'vue-router'

const originalPush = Router.prototype.push

Router.prototype.push = function push(location) {

  return originalPush.call(this, location).catch(err => err)

}

## Vue中bus的使用

安装：npm install vue-bus

在main.js中引入vue-bus

import Vue from 'vue';

import VueBus from 'vue-bus';

Vue.use(VueBus);

1，需要先新建一个bus.js文件，并写入

import Vue from 'vue';

// 使用 Event Bus

const bus = new Vue();

export default bus;

2，在接收组件和传递组件中分别引入该文件

3，在组件中使用bus

// A组件触发事件(传递数据)：

this.$bus.emit("parameterName",params)；

// B组件接收事件()：

this.$bus.on('parameterName', function (value) {

    console.log(value);

});

// 组件销毁时解除事件绑定：

destroyed:function(){

　　this.$bus.off("parameterName")

}

## CSS渐变

字体渐变

.fontcur0 {

    background-image: linear-gradient(135deg, #1099d0, #41dcd6);

    font-weight: bold;

    -webkit-background-clip: text;

    -moz-background-clip: text;

    background-clip: text;

    box-decoration-break: clone;

    -webkit-box-decoration-break: clone;

    -moz-box-decoration-break: clone;

    color: transparent;

    position: relative;

}

背景渐变

/\* 背景渐变 \*/

.cur0 {

    background-image: linear-gradient(#1099d0, #41dcd6);

}

.cur1 {

    background-image: linear-gradient(#966ffa, #ff4c99);

}

.cur2 {

    background-image: linear-gradient(#f3646c, #f6c988);

}

.cur3 {

    background-image: linear-gradient(#7956fd, #a5c6fe);

}

.cur4 {

    background-image: linear-gradient(#ffab96, #ff0f47);

}

## Flex布局

**一：以下6个属性设置在容器上（父元素）**

* flex-direction row（默认值）：主轴为水平方向，起点在左端。

row-reverse：主轴为水平方向，起点在右端。

row-reverse：主轴为水平方向，起点在右端。

column：主轴为垂直方向，起点在上沿。

column-reverse：主轴为垂直方向，起点在下沿。

* flex-wrap nowrap（默认）：不换行。

wrap：换行，第一行在上方

wrap-reverse：换行，第一行在下方

* flex-flow 简写：有两个参数（flex-dicration，flex-wrap）
* justify-content（水平）：flex-start | flex-end | center | space-between | space-around（每个项目两侧的间隔相等）;
* align-items（垂直）：flex-start（顶部对齐） | flex-end（底部对齐） | center | baseline（文字基线对齐） | stretch（（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度）;
* align-content（属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用）

flex-start ：与交叉轴的起点对齐

flex-end ：与交叉轴的终点对齐

center ：与交叉轴的中点对齐

space-between ：与交叉轴两端对齐，轴线之间的间隔平均分布

space-around ：每根轴线两侧的间隔都相等

stretch（默认值）：轴线占满整个交叉轴

**二：项目属性（子元素）**

* order：定义项目的排列顺序，数值越小，排列越靠前，默认为0
* flex-grow：定义项目的放大比例，默认为0，即如果存在剩余空间，也不放大
* flex-shrink：定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小，

如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小，

如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，

前者不缩小，负值对该属性无效

* flex-basis：定义了在分配多余空间之前，项目占据的主轴空间（main size），浏览器根据这个属性，计算主轴是 否有多余空间，它的默认值为auto，即项目的本来大小
* flex：是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto，后两个属性可选
* align-self：auto | flex-start | flex-end | center | baseline | stretch；

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性，默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch

该属性可能取6个值，除了auto，其他都与align-items属性完全一致

## BUS的使用

使用：兄弟组件之间进行传值；

安装：npm install vue-bus

在main.js中引入vue-bus

 import Vue from 'vue';

 import VueBus from 'vue-bus';

 Vue.use(VueBus);

在组件中使用：

A组件触发事件：需要在menthods中定义，在mounted中调用

bus.$emit("parameterName", params)；

B组件接收事件：在created生命周期中触发（否则数据在页面刷新后悔丢失）

bus.$on("parameterName", function(value) {

  console.log(value)

});

 组件销毁时解除事件绑定：

destroyed: function() {

bus.off("parameterName")

}

## watch的使用

1. 直接写一个监听处理函数，当每次监听到 cityName 值发生改变时，执行函数（普通监听方式），这样使用watch时有一个特点，就是当值第一次绑定的时候，不会执行监听函数，只有值发生改变才会执行。如果我们需要在最初绑定值的时候也执行函数，则就需要用到immediate属性

export default {

    data() {

        return {

            cityName: '' //直接写一个监听处理函数，当每次监听到 cityName 值发生改变时，执行函数

        };

    },

    watch: {

        cityName(newName, oldName) {

            // ...

        }

    }

};

2. **immediate和handler**

比如当父组件向子组件动态传值时，子组件props首次获取到父组件传来的默认值时，也需要执行函数，此时就需要将immediate设为true，immediate表示在watch中首次绑定的时候，是否执行handler，值为true则表示在watch中声明的时候，就立即执行handler方法，值为false，则和一般使用watch一样，在数据发生变化的时候才执行handler

    watch: {

        $route: {

            handler(newvalue, oldvalue) {//hander不能换名

                console.log(newvalue);

            },

            immediate: true

        }

    },

3. **deep**

当需要监听一个对象的改变时，普通的watch方法无法监听到对象内部属性的改变，只有data中的数据才能够监听到变化，此时就需要deep属性对对象进行深度监听。

    watch: {

        cityName: {

            handler(newName, oldName) {

                // ...

            },

            deep: true,

            immediate: true

        }

},

**Vue 子组件与父组件之间的双向数据绑定**

1、子父，父子传值中数据同步更新方法

父组件中

<tablea ref="tablea" v-if="Datar0 != ''" :pageid="pageid" :Datar0="Datar0"></tablea>

    watch: {

        Datar0: {

            handler(newdata, oldata) {

                this.Datar0 = newdata;

            },

            deep: true,

            immediate: true

        }

},

2、其中Datar0为父组件传递给子组件的值，使用监听器监听其值的变化，并将新值传递给子组件

子组件中同样也需要监听父组件传递过去的值，并将新值（也即是修改过后的值渲染到页面），要点就是子父组件监听同一个值就可完成数据的双向同步

    watch: {

        Datar0: {

            handler(newdata, oldata) {

                this.data6 = newdata;

            },

            deep: true,

            immediate: true

        }

    },

## 路由拦截器

**一、全局路由拦截器**

① 在 src 目录下创建一个 interceptors 文件夹，

interceptors 文件夹下创建一个 index.js 文件，用于路由拦截的代码编辑。

② 全局引入 interceptors /index.js

在 main.js 中引入 interceptor

// 全局引入路由拦截器

import './interceptors/index.js';

**编辑 interceptors**

1. 引入 aixos，router

是给路由设置的，一定少不了 router，请求拦截的，少不了 axios

import axios from 'axios'

import Router from 'vue-router';

1. 代码编写

router.beforeEach((to, from, next) => {

    if (to.meta.requireAuth) { // 判断该路由是否需要登录权限

        if (store.state.token) { // 通过vuex state获取当前的token是否存在

            next();

        } else {

            next({

                path: '/login',

                query: {

                    redirect: to.fullPath

                } // 将跳转的路由path作为参数，登录成功后跳转到该路由

            })

        }

    } else {

        next();

    }

})

**二、局部路由拦截器**

路由拦截器就是路由守卫，分为全局守卫和局部守卫

**1、全局守卫写在路由定义的文件里面，每当路由切换就会执行**

// 进入页面之前执行

router.beforeEach((to, from, next) => {

  // ...

})

// 进入页面之后执行

router.afterEach((to, from) => {

  // ...

})

**2、局部守卫写在vue组件里面，只对当前组件生效，和vue生命周期函数写法一样**

    beforeRouteEnter(to, from, next) {

        // 在渲染该组件的对应路由被 confirm 前调用

        // 不！能！获取组件实例 `this`

        // 因为当守卫执行前，组件实例还没被创建

    },

    beforeRouteUpdate(to, from, next) {

        // 在当前路由改变，但是该组件被复用时调用

        // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

        // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

        // 可以访问组件实例 `this`

    },

    beforeRouteLeave(to, from, next) {

        // 导航离开该组件的对应路由时调用

        // 可以访问组件实例 `this`

    }

**每个守卫方法接收三个参数：**

to: 即将要进入的目标 路由对象

from: 当前导航正要离开的路由

next:一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。

next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 confirmed (确认的)。

next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL地址会重置到 from 路由对应的地址。

next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 router-link 的 to prop 或 router.push 中的选项。

next(error): (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 router.onError() 注册过的回调。

\*\*确保 next 函数在任何给定的导航守卫中都被严格调用一次。它可以出现多于一次，但是只能在所有的逻辑路径都不重叠的情况下，否则钩子永远都不会被解析或报错。\*\*

## JS将数组转换成字符串

            let newidData = this.dataIder.join(',');

## 父组件调用子组件中的function

子组件中定义

    methods: {

        teatData() {

            this.dwrowimg = this.Datar0;

        },

}

父组件

<tablea ref="tablea" v-if="Datar0 != ''" :pageid="pageid" :Datar0="Datar0"></tablea>

调用方法

        query() {

            this.$refs.tablea.teatData();

        },

This.$refs.子组件名称的ref值.方法名称（）

## 子组件调用父组件中的function

**第一种：**

父组件定义

        forChild() {

            alert('asdasdasdasdasdasdasdasd');

        },

子组件调用

        remove(index) {

            if (this.pageId == 20) {

                this.$parent.forChild();

            }

        },

可以传参

**第二种：**

使用this.$emit()调用

子组件定义

        remove(index) {

            if (this.pageId == 20) {

                this.$emit('bbb', '我是小鱼儿');

                this.data6.splice(index, 1);

            }

        },

// 在子组件上绑定bbb

<tablea @bbb="forChild" v-if="Datar20 != ''" :pageid="pageid" :Datar20="Datar20"></tablea>

父组件定义

        forChild(aaa) {

            alert(aaa);

        },

**第三种：**

父组件中定义自定义属性，将该方法传递到子组件中，子组件中通过props接收，再进行调用，这里也可以进行传值

 <tablea :bbb="forChild" v-if="Datar20 != ''" :pageid="pageid" :Datar20="Datar20"></tablea>

        // 相当于吧forChild这个函数当成值传递

forChild(aaa) {

            alert(aaa);

        },

子组件接收

    props: [

        'bbb',

]

子组件接收后进行传值调用

        remove(index) {

            if (this.pageId == 20) {

                if (this.bbb) {

                    this.bbb(123);

                }

            }

        },

## Vue在JS文件中暴露函数，在组件中接收调用函数的方法

**创建JS文件**

function myFunction(data) { //定义函数

    console.log(data);

    console.log('hsaduisahdui');

}

export {

    myFunction //暴露函数

}

**在组件中接收**

import { myFunction } from '../../../xiaoyuer/xiaoyuer.js';

**调用方法**

        dayuer() {

            myFunction();

        },

## CSS选中基数项和偶数项

li:nth-child(odd) { /\* 基数项 \*/

    background-color: red;

}

li:nth-child(even) { /\* 偶数项 \*/

    background-color: red;

}

## JavaScript 正则表达式

**语法**

/正则表达式主体/修饰符(可选)

其中修饰符是可选的。

实例：

var patt = /runoob/i

实例解析：

**/runoob/i**  是一个正则表达式。

**runoob**  是一个**正则表达式主体** (用于检索)。

**i**  是一个**修饰符** (搜索不区分大小写)。

**使用字符串方法**

**在 JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。**

**search() 方法** 用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子串的起始位置。

**replace() 方法** 用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

**search() 方法使用正则表达式**

实例

使用正则表达式搜索 "Runoob" 字符串，且不区分大小写：

var str = "Visit Runoob!"; var n = str.search(/Runoob/i);

输出结果为：

6

**search() 方法使用字符串**

search 方法可使用字符串作为参数。字符串参数会转换为正则表达式：

实例

检索字符串中 "Runoob" 的子串：

var str = "Visit Runoob!"; var n = str.search("Runoob");

**replace() 方法使用正则表达式**

实例

使用正则表达式且不区分大小写将字符串中的 Microsoft 替换为 Runoob :

var str = document.getElementById("demo").innerHTML; var txt = str.replace(/microsoft/i,"Runoob");

结果输出为:

Visit Runoob!

**replace() 方法使用字符串**

replace() 方法将接收字符串作为参数：

var str = document.getElementById("demo").innerHTML; var txt = str.replace("Microsoft","Runoob");

**正则表达式修饰符**

**修饰符** 可以在全局搜索中不区分大小写:

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| i | 执行对大小写不敏感的匹配。 |
| g | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

**正则表达式模式**

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| (x|y) | 查找任何以 | 分隔的选项。 |

元字符是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| \d | 查找数字。 |
| \s | 查找空白字符。 |
| \b | 匹配单词边界。 |
| \uxxxx | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

量词:

|  |  |
| --- | --- |
| **量词** | **描述** |
| n+ | 匹配任何包含至少一个 n 的字符串。 |
| n\* | 匹配任何包含零个或多个 n 的字符串。 |
| n? | 匹配任何包含零个或一个 n 的字符串。 |

**使用 RegExp 对象**

在 JavaScript 中，RegExp 对象是一个预定义了属性和方法的正则表达式对象。

**使用 test()**

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

实例

var patt = /e/;  
patt.test("The best things in life are free!");

字符串中含有 "e"，所以该实例输出为：

true

你可以不用设置正则表达式的变量，以上两行代码可以合并为一行：

/e/.test("The best things in life are free!")

**使用 exec()**

exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":实例 1

/e/.exec("The best things in life are free!");

字符串中含有 "e"，所以该实例输出为:E

## js 中 try catch用法

try {

                tryStatements; // 必选项，可能发生错误的语句

            } catch (exception) {// 必选项，任何变量名，exception的初始化值是扔出的错误的值

                catchStatements; // 可选项,处理在相关联的，tryStatement中发生的错误的语句

            } finally {

                finallyStatements; // 可选项，在所有其他过程发生之后无条件执行的语句

            }

## render函数在view中的用法

render: (h, params) => {

    return h(

        'div',

        {

            attrs: {

                style: 'width:100%; height: auto;display:block;padding-top:15px;padding-bottom: 15px;'

            }

        },

        [

            h('img', {

                props: {

                    type: 'primary',

                    size: 'small'

                },

                attrs: {

                    src: params.row.col5,

                    style: 'width:100%; height: auto;'

                }

            })

        ]

    );

}

## CSS修改placeholder的样式

input::-webkit-input-placeholder {

    color: #aab2bd;

    font-size: 12px;

    text-align: left;

}

## JS校验密码复杂度（必须包含字母、数字、特殊符号）

### #场景一：密码中必须包含大小写 字母、数字、特称字符，至少8个字符，最多30个字符；

 var pwdRegex = new RegExp('(?=.\*[0-9])(?=.\*[A-Z])(?=.\*[a-z])(?=.\*[^a-zA-Z0-9]).{8,30}');

       if (!pwdRegex.test('A3b@C2dEF')) {

            alert('您的密码复杂度太低（密码中必须包含大小写字母、数字、特殊字符），请及时修改密码！');

       }

### #场景二：密码中必须包含字母（不区分大小写）、数字、特称字符，至少8个字符，最多30个字符；

            var pwdRegex = new RegExp('(?=.\*[0-9])(?=.\*[a-zA-Z])(?=.\*[^a-zA-Z0-9]).{8,30}');

            if (!pwdRegex.test('a2b3c$def')) {

                alert('您的密码复杂度太低（密码中必须包含字母、数字、特殊字符），请及时修改密码！');

            }

### #场景三：密码中必须包含字母（不区分大小写）、数字，至少8个字符，最多30个字符；

            var pwdRegex = new RegExp('(?=.\*[0-9])(?=.\*[a-zA-Z]).{8,30}');

            if (!pwdRegex.test('a2b3c4d5')) {

                alert('您的密码复杂度太低（密码中必须包含字母、数字），请及时修改密码！');

            }

## CSS的三元表达式

一：使用type

<div :type="tagData === 'all' ? '' : 'warning'" effect="dark" @click="toggleTag('all')">全部</div>

<div :type="tagData === 'undone' ? '' : 'warning' " @click="toggleTag('undone')" effect="dark">未完成</div>

<div :type="tagData === 'done' ? '' : 'warning' " @click="toggleTag('done')" effect="dark">完成</div>

二使用class

<div:class="i==0?'sonClassDicvxiaoyuer':'sonClassDicvxiaoyuer2'" >{{ item.percentage }}</div>

## CSS的三元表达式的嵌套使用

<div  :class="i==0?'sonClassDicvxiaoyuer':i==1?'sonClassDicvxiaoyuer2':'sonClassDicvxiaoyuer3'">{{ item.percentage }}</div>

## JS获取标签中的值

                        if (i == j) {

                            Percentage[i].style.width = this.ProgressBar[j];

                            console.log(Percentage[i].innerText); // 获取标签中的值

                        }

## CSS的三元表达式

监听事件的绑定与移除主要是addEventListener和removeEventListener的运用。

## ddEventListener语法

element.addEventListener(type，handler，false/true)

type:事件类型

handler:事件执行触发的函数

false/true:false为冒泡/ture为捕获，参数是true，表示在捕获阶段调用事件处理程序；如果是false，表示在冒泡阶段调用事件处理程序。

事件捕获：父级元素先触发，子集元素后触发；

事件冒泡：子集元素先触发，父级元素后触发；

一般的绑定事件，都是采用冒泡方式，也就是使用false

## removeEventListener语法

element.removeEventListener(type，handler，false/true)

## JS获取元素的大小（高度和宽度）

|  |  |
| --- | --- |
| 与元素尺寸相关的属性 | |
| **元素尺寸属性** | **说明** |
| clientWidth | 获取元素可视部分的宽度，即 CSS 的 width 和 padding 属性值之和，元素边框和滚动条不包括在内，也不包含任何可能的滚动区域 |
| clientHeight | 获取元素可视部分的高度，即 CSS 的 height 和 padding 属性值之和，元素边框和滚动条不包括在内，也不包含任何可能的滚动区域 |
| offsetWidth | 元素在页面中占据的宽度总和，包括 width、padding、border 以及滚动条的宽度 |
| offsetHeight | 元素在页面中占据的高度总和，包括 height、padding、border 以及滚动条的宽度 |
| scrollWidth | 当元素设置了 overflow:visible 样式属性时，元素的总宽度，也称滚动宽度。在默认状态下，如果该属性值大于 clientWidth 属性值，则元素会显示滚动条，以便能够翻阅被隐藏的区域 |
| scrollHeight | 当元素设置了 overflow:visible 样式属性时，元素的总高度，也称滚动高度。在默认状态下，如果该属性值大于 clientWidth 属性值，则元素会显示滚动条，以便能够翻阅被隐藏的区域 |

## 统计图基础

Html

<div ref="main02" style="width: 100%; height:400px"></div>

Script

var echarts = require('echarts');

methods: {

        xiaoyuerEcharse() {

            var myChart = echarts.init(this.$refs.main02);

            // 指定图表的配置项和数据

            var option = {

                xAxis: {

                    type: 'category',

                    data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

                },

                yAxis: {

                    type: 'value'

                },

                series: [

                    {

                        data: [150, 230, 224, 218, 135, 147, 260],

                        type: 'line',

                        color: 'rgb(0, 187, 122)'

                    }

                ]

            };

            // 使用刚指定的配置项和数据显示图表。

            myChart.setOption(option);

        }

    }

## echart配置标题

var option = {

        //标题

        title: {

            show: true, //显示策略，默认值true,可选为：true（显示） | false（隐藏）

            text: '主标题', //主标题文本，'\n'指定换行

            link: '', //主标题文本超链接,默认值true

            target: null, //指定窗口打开主标题超链接，支持'self' | 'blank'，不指定等同为'blank'（新窗口）

            subtext: '副标题', //副标题文本，'\n'指定换行

            sublink: '', //副标题文本超链接

            subtarget: null, //指定窗口打开副标题超链接，支持'self' | 'blank'，不指定等同为'blank'（新窗口）

            x: 'center' //水平安放位置，默认为'left'，可选为：'center' | 'left' | 'right' | {number}（x坐标，单位px）

            y: 'top', //垂直安放位置，默认为top，可选为：'top' | 'bottom' | 'center' | {number}（y坐标，单位px）

            textAlign: null, //水平对齐方式，默认根据x设置自动调整，可选为： left' | 'right' | 'center

            backgroundColor: 'rgba(0,0,0,0)', //标题背景颜色，默认'rgba(0,0,0,0)'透明

            borderColor: '#ccc', //标题边框颜色,默认'#ccc'

            borderWidth: 0, //标题边框线宽，单位px，默认为0（无边框）

            padding: 5, //标题内边距，单位px，默认各方向内边距为5，接受数组分别设定上右下左边距

            itemGap: 10, //主副标题纵向间隔，单位px，默认为10

            textStyle: { //主标题文本样式{"fontSize": 18,"fontWeight": "bolder","color": "#333"}

                fontFamily: 'Arial, Verdana, sans...',

                fontSize: 12,

                fontStyle: 'normal',

                fontWeight: 'normal',

            },

            subtextStyle: { //副标题文本样式{"color": "#aaa"}

                fontFamily: 'Arial, Verdana, sans...',

                fontSize: 12,

                fontStyle: 'normal',

                fontWeight: 'normal',

            },

            zlevel: 0, //一级层叠控制。默认0,每一个不同的zlevel将产生一个独立的canvas，相同zlevel的组件或图标将在同一个canvas上渲染。zlevel越高越靠顶层，canvas对象增多会消耗更多的内存和性能，并不建议设置过多的zlevel，大部分情况可以通过二级层叠控制z实现层叠控制。

            z: 6, //二级层叠控制，默认6,同一个canvas（相同zlevel）上z越高约靠顶层。

        },

## Echarts 柱状图配置详解

**所有基础配置**

// 指定图表的配置项和数据

var option = {

    // ----  标题 -----

    title: {

        text: '主标题',

        textStyle: {

            color: 'red'

        },

        subtext: '副标题',

        subtextStyle: {

            color: 'blue'

        },

        padding: [0, 0, 10, 100] // 位置

    },

    // ---- legend ----

    legend: {

        type: 'plain', // 图列类型，默认为 'plain'

        top: '1%', // 图列相对容器的位置 top\bottom\left\right

        selected: {

            // '销量': true  // 图列选择，图形加载出来会显示选择的图列，默认为true

        },

        textStyle: { // 图列内容样式

            color: '#fff', // 字体颜色

            backgroundColor: 'black' // 字体背景色

        },

        tooltip: { // 图列提示框，默认不显示

            show: true,

            color: 'red'

        },

        data: [ // 图列内容

            {

                name: '销量',

                icon: 'circle',

                textStyle: {

                    color: 'red', // 单独设置某一个图列的颜色

                    backgroundColor: '#fff' // 单独设置某一个图列的字体背景色

                }

            }

        ]

    },

    // ---  提示框 ----

    tooltip: {

        show: true, // 是否显示提示框，默认为true

        trigger: 'item', // 数据项图形触发

        axisPointer: { // 指示样式

            type: 'shadow',

            axis: 'auto'

        },

        padding: 5,

        textStyle: { // 提示框内容的样式

            color: '#fff'

        }

    },

    // ---- gird区域 ---

    gird: {

        show: false, // 是否显示直角坐标系网格

        top: 80, // 相对位置 top\bottom\left\right

        containLabel: false, // gird 区域是否包含坐标轴的刻度标签

        tooltip: {

            show: true,

            trigger: 'item', // 触发类型

            textStyle: {

                color: '#666'

            }

        }

    },

    //  ------  X轴 ------

    xAxis: {

        show: true, // 是否显示

        position: 'bottom', // x轴的位置

        offset: 0, // x轴相对于默认位置的偏移

        type: 'category', // 轴类型， 默认为 'category'

        name: '月份', // 轴名称

        nameLocation: 'end', // 轴名称相对位置

        nameTextStyle: { // 坐标轴名称样式

            color: 'red',

            padding: [5, 0, 0, -5]

        },

        nameGap: 15, // 坐标轴名称与轴线之间的距离

        nameRotate: 0, // 坐标轴名字旋转

        axisLine: { // 坐标轴 轴线

            show: true, // 是否显示

            symbol: ['none', 'arrow'], // 是否显示轴线箭头

            symbolSize: [8, 8], // 箭头大小

            symbolOffset: [0, 7], // 箭头位置

            // ------   线 ---------

            lineStyle: {

                color: 'blue',

                width: 1,

                type: 'solid'

            }

        },

        axisTick: { // 坐标轴 刻度

            show: true, // 是否显示

            inside: true, // 是否朝内

            length: 3, // 长度

            lineStyle: { // 默认取轴线的样式

                color: 'red',

                width: 1,

                type: 'solid'

            }

        },

        axisLabel: { // 坐标轴标签

            show: true, // 是否显示

            inside: false, // 是否朝内

            rotate: 0, // 旋转角度

            margin: 5, // 刻度标签与轴线之间的距离

            color: 'red' // 默认取轴线的颜色

        },

        splitLine: { // gird区域中的分割线

            show: false, // 是否显示

            lineStyle: {

                // color: 'red',

                // width: 1,

                // type: 'solid'

            }

        },

        splitArea: { // 网格区域

            show: false // 是否显示，默认为false

        },

        data: ['1月', '2月', '3月', '4月', '5月', '6月', '7月', '8月', '9月', '10月', '11月', '12月']

    },

    //   ------   y轴  ----------

    yAxis: {

        show: true, // 是否显示

        position: 'left', // y轴位置

        offset: 0, // y轴相对于默认位置的偏移

        type: 'value', // 轴类型，默认为 ‘category’

        name: '销量', // 轴名称

        nameLocation: 'end', // 轴名称相对位置value

        nameTextStyle: { // 坐标轴名称样式

            color: '#fff',

            padding: [5, 0, 0, 5] // 坐标轴名称相对位置

        },

        nameGap: 15, // 坐标轴名称与轴线之间的距离

        nameRotate: 270, // 坐标轴名字旋转

        axisLine: { // 坐标轴 轴线

            show: true, // 是否显示

            //  -----   箭头 -----

            symbol: ['none', 'arrow'], // 是否显示轴线箭头

            symbolSize: [8, 8], // 箭头大小

            symbolOffset: [0, 7], // 箭头位置

            // ----- 线 -------

            lineStyle: {

                color: 'blue',

                width: 1,

                type: 'solid'

            }

        },

        axisTick: { // 坐标轴的刻度

            show: true, // 是否显示

            inside: true, // 是否朝内

            length: 3, // 长度

            lineStyle: {

                color: 'red', // 默认取轴线的颜色

                width: 1,

                type: 'solid'

            }

        },

        axisLabel: { // 坐标轴的标签

            show: true, // 是否显示

            inside: false, // 是否朝内

            rotate: 0, // 旋转角度

            margin: 8, // 刻度标签与轴线之间的距离

            color: 'red', // 默认轴线的颜色

        },

        splitLine: { // gird 区域中的分割线

            show: true, // 是否显示

            lineStyle: {

                color: '#666',

                width: 1,

                type: 'dashed'

            }

        },

        splitArea: { // 网格区域

            show: false // 是否显示，默认为false

        }

    },

    //  -------   内容数据 -------

    series: [{

        name: '销量', // 序列名称

        type: 'bar', // 类型

        legendHoverLink: true, // 是否启用图列 hover 时的联动高亮

        label: { // 图形上的文本标签

            show: false,

            position: 'insideTop', // 相对位置

            rotate: 0, // 旋转角度

            color: '#eee'

        },

        itemStyle: { // 图形的形状

            color: 'blue',

            barBorderRadius: [18, 18, 0, 0]

        },

        barWidth: 20, // 柱形的宽度

        barCategoryGap: '20%', // 柱形的间距

        data: [3000, 4000, 4200, 4500, 6000, 5600, 4500, 5020, 4500, 5400, 4300, 1200]

    }]

};

**HTML配置**

< div id="main" style="width: 600px;height:400px;margin: 0 auto" > < /div>

 <script type = "text/javascript" >

        // 基于准备好的dom，初始化echarts实例

        var myChart = echarts.init(document.getElementById('main'));

        // 使用刚指定的配置项和数据显示图表。

        myChart.setOption(option);

 < /script>

## Echarts折叠柱状图配置

**所有基础配置**

option = {

    title: {

        text: "",

        textStyle: {

            color: "#436EEE",

            fontSize: 17

        }

    },

    tooltip: {

        trigger: 'axis',

        axisPointer: {

            type: 'shadow'

        }

    },

    legend: {

        data: [{

                name: '政策法规',

                icon: 'circle',

                textStyle: {

                    color: 'red', // 单独设置某一个图列的颜色

                    backgroundColor: '#fff' // 单独设置某一个图列的字体背景色

                }

            },

            {

                name: '经办规程',

                icon: 'circle',

                textStyle: {

                    color: 'red', // 单独设置某一个图列的颜色

                    backgroundColor: '#fff' // 单独设置某一个图列的字体背景色

                }

            },

            {

                name: '业务场景模拟',

                icon: 'circle',

                textStyle: {

                    color: 'red', // 单独设置某一个图列的颜色

                    backgroundColor: '#fff' // 单独设置某一个图列的字体背景色

                }

            },

            {

                name: '常见问题',

                icon: 'circle',

                textStyle: {

                    color: 'red', // 单独设置某一个图列的颜色

                    backgroundColor: '#fff' // 单独设置某一个图列的字体背景色

                }

            }

        ]

    },

    //x轴显示

    xAxis: {

        data: [20180611, 20180612, 20180613, 20180614, 20180615, 20180616, 20180617],

        show: true, // 是否显示

        position: 'bottom', // x轴的位置

        offset: 0, // x轴相对于默认位置的偏移

        type: 'category', // 轴类型， 默认为 'category'

        name: '月份', // 轴名称

        nameLocation: 'end', // 轴名称相对位置

        nameTextStyle: { // 坐标轴名称样式

            color: 'red',

            padding: [5, 0, 0, -5]

        },

        nameGap: 15, // 坐标轴名称与轴线之间的距离

        nameRotate: 0, // 坐标轴名字旋转

        axisLine: { // 坐标轴 轴线

            show: true, // 是否显示

            symbol: ['none', 'arrow'], // 是否显示轴线箭头

            symbolSize: [8, 8], // 箭头大小

            symbolOffset: [0, 7], // 箭头位置

            // ------   线 ---------

            lineStyle: {

                color: 'blue',

                width: 1,

                type: 'solid'

            }

        },

        axisTick: { // 坐标轴 刻度

            show: true, // 是否显示

            inside: true, // 是否朝内

            length: 3, // 长度

            lineStyle: { // 默认取轴线的样式

                color: 'red',

                width: 1,

                type: 'solid'

            }

        },

        axisLabel: { // 坐标轴标签

            show: true, // 是否显示

            inside: false, // 是否朝内

            rotate: 0, // 旋转角度

            margin: 5, // 刻度标签与轴线之间的距离

            color: 'red' // 默认取轴线的颜色

        },

        splitLine: { // gird区域中的分割线

            show: false, // 是否显示

            lineStyle: {

                // color: 'red',

                // width: 1,

                // type: 'solid'

            }

        },

        splitArea: { // 网格区域

            show: false // 是否显示，默认为false

        },

        // show: false

    },

    //   ------   y轴  ----------

    yAxis: {

        show: true, // 是否显示

        position: 'left', // y轴位置

        offset: 0, // y轴相对于默认位置的偏移

        type: 'value', // 轴类型，默认为 ‘category’

        name: '销量', // 轴名称

        nameLocation: 'end', // 轴名称相对位置value

        nameTextStyle: { // 坐标轴名称样式

            color: '#fff',

            padding: [5, 0, 0, 5] // 坐标轴名称相对位置

        },

        nameGap: 15, // 坐标轴名称与轴线之间的距离

        nameRotate: 270, // 坐标轴名字旋转

        axisLine: { // 坐标轴 轴线

            show: true, // 是否显示

            //  -----   箭头 -----

            symbol: ['none', 'arrow'], // 是否显示轴线箭头

            symbolSize: [8, 8], // 箭头大小

            symbolOffset: [0, 7], // 箭头位置

            // ----- 线 -------

            lineStyle: {

                color: 'blue',

                width: 1,

                type: 'solid'

            }

        },

        axisTick: { // 坐标轴的刻度

            show: true, // 是否显示

            inside: true, // 是否朝内

            length: 3, // 长度

            lineStyle: {

                color: 'red', // 默认取轴线的颜色

                width: 1,

                type: 'solid'

            }

        },

        axisLabel: { // 坐标轴的标签

            show: true, // 是否显示

            inside: false, // 是否朝内

            rotate: 0, // 旋转角度

            margin: 8, // 刻度标签与轴线之间的距离

            color: 'red', // 默认轴线的颜色

        },

        splitLine: { // gird 区域中的分割线

            show: true, // 是否显示

            lineStyle: {

                color: '#666',

                width: 1,

                type: 'dashed'

            }

        },

        splitArea: { // 网格区域

            show: false // 是否显示，默认为false

        }

    },

    tooltip: {

        position: ['50%', '50%'],

        padding: [5, 10]

    },

    grid: {

        left: '0%'

    },

    series: [{

            name: "政策法规",

            type: "bar",

            stack: "业务", //折叠显示

            data: ["87.54", "88.54", "90", "91", "92", '95', '100'],

            barWidth: 25,

            //显示颜色

            itemStyle: {

                normal: { color: "blue" }

            },

            label: { // 图形上的文本标签

                show: false,

                position: 'insideTop', // 相对位置

                rotate: 0, // 旋转角度

                color: '#eee'

            }

        },

        {

            name: "经办规程",

            type: "bar",

            stack: "业务",

            data: ["87.54", "88.54", "90", "91", "92", '95', '100'],

            barWidth: 25,

            label: { // 图形上的文本标签

                show: false,

                position: 'insideTop', // 相对位置

                rotate: 0, // 旋转角度

                color: '#eee'

            },

            itemStyle: {

                normal: { color: "#FF8849" }

            }

        },

        {

            name: "业务场景模拟",

            type: "bar",

            stack: "业务",

            data: ["87.54", "88.54", "90", "91", "92", '95', '100'],

            barWidth: 25,

            label: { // 图形上的文本标签

                show: false,

                position: 'insideTop', // 相对位置

                rotate: 0, // 旋转角度

                color: '#eee'

            },

            itemStyle: {

                normal: { color: "#3FBB49" }

            }

        },

        {

            name: "常见问题",

            type: "bar",

            stack: "业务",

            data: ["87.54", "88.54", "90", "91", "92", '95', '100'],

            barWidth: 25,

            label: { // 图形上的文本标签

                show: false,

                position: 'insideTop', // 相对位置

                rotate: 0, // 旋转角度

                color: '#eee'

            },

            itemStyle: {

                normal: { color: "#56C4A5" }

            },

            barCateGoryGap: 15

        }

    ]

};

**HTML配置**

    <!-- 为ECharts准备一个具备大小（宽高）的Dom -->

    <div id = "main" style="width: 600px;height:400px;margin: 0 auto" > < /div>

    <script type = "text/javascript" >

        // 基于准备好的dom，初始化echarts实例

        var myChart = echarts.init(document.getElementById('main'));

    // 使用刚指定的配置项和数据显示图表。

    myChart.setOption(option);

    myChart.on('click', function(params) {

        // 控制台打印数据的名称

        console.log(params);

    });

    </script>

## JS区域拖动

<template>

    <div id="box1">

        <div id="box"></div>

    </div>

</template>

<script>

export default {

    mounted() {

        this.xiaoyuerSwiper();

    },

    methods: {

        // 布局函数

        xiaoyuerSwiper() {

            (function () {

                var dragging = false;

                var boxX, boxY, mouseX, mouseY, offsetX, offsetY;

                var box = document.getElementById('box');

                var box1 = document.getElementById('box1');

                // 鼠标按下的动作

                box.onmousedown = down;

                // 鼠标的移动动作

                document.onmousemove = move;

                // 释放鼠标的动作

                document.onmouseup = up;

                // 鼠标按下后的函数,e为事件对象

                function down(e) {

                    dragging = true;

                    // 获取元素所在的坐标

                    boxX = box.offsetLeft;

                    boxY = box.offsetTop;

                    // 获取鼠标所在的坐标

                    mouseX = parseInt(getMouseXY(e).x);

                    mouseY = parseInt(getMouseXY(e).y);

                    // 鼠标相对元素左和上边缘的坐标

                    offsetX = mouseX - boxX;

                    offsetY = mouseY - boxY;

                }

                // 鼠标移动调用的函数

                function move(e) {

                    if (dragging) {

                        // 获取移动后的元素的坐标

                        var x = getMouseXY(e).x - offsetX;

                        var y = getMouseXY(e).y - offsetY;

                        // 计算可移动位置的大小， 保证元素不会超过可移动范围

                        // 此处就是父元素的宽度减去子元素宽度

                        var width = box1.clientWidth - box.offsetWidth;

                        var height = box1.clientHeight - box.offsetHeight;

                        // min方法保证不会超过右边界，max保证不会超过左边界

                        x = Math.min(Math.max(0, x), width);

                        y = Math.min(Math.max(0, y), height);

                        // 给元素及时定位

                        box.style.left = x + 'px';

                        box.style.top = y + 'px';

                    }

                }

                // 释放鼠标的函数

                function up(e) {

                    dragging = false;

                }

                // 函数用于获取鼠标的位置

                function getMouseXY(e) {

                    var x = 0,

                        y = 0;

                    e = e || window.event;

                    if (e.pageX) {

                        x = e.pageX;

                        y = e.pageY;

                    } else {

                        x = e.clientX + document.body.scrollLeft - document.body.clientLeft;

                        y = e.clientY + document.body.scrollTop - document.body.clientTop;

                    }

                    return {

                        x: x,

                        y: y

                    };

                }

            })();

        }

    }

};

</script>

<style scope="this api replaced by slot-scope in 2.5.0+">

/\* right \*/

#box1 {

    width: 500px;

    height: 100px;

    background: #999;

    position: relative;

    left: 100px;

    top: 100px;

}

#box {

    width: 100px;

    height: 100px;

    background: #334;

    position: absolute;

    cursor: move;

}

</style>

## CSS单行文字溢出省略号（溢出隐藏）

.YuyuanGuard\_2 {/\*\*单行溢出影藏 \*/

    height: 42px;

    overflow: hidden;

    text-overflow: ellipsis;

    white-space: nowrap;

}

.YuyuanGuard\_2 {/\*\*多行溢出影藏 \*/

    height: 42px;

    display: -webkit-box;

    -webkit-box-orient: vertical;

    -webkit-line-clamp: 2;

    text-overflow: ellipsis;

    overflow: hidden;

}

## CSS滚动条样式

.TextClassStyle::-webkit-scrollbar {

    /\*\*滚动条整体部分 \*/

    width: 10px;

    height: 10px;

}

.TextClassStyle::-webkit-scrollbar-track {

    /\*\*滚动条里面的小方块，能向上向下移动（或往左往右移动，取决于是垂直滚动条还是水平滚动条） \*/

    background: rgb(239, 239, 239);

    border-radius: 2px;

}

.TextClassStyle::-webkit-scrollbar-thumb {

    /\*\*滚动条的轨道（里面装有Thumb） \*/

    background: #bfbfbf;

    border-radius: 10px;

}

.TextClassStyle::-webkit-scrollbar-thumb:hover {

    /\* \*/

    background: #333;

}

.TextClassStyle::-webkit-scrollbar-corner {

    /\*\* \*/

    background: #179a16;

}

**说明：**

::-webkit-scrollbar 滚动条整体部分

::-webkit-scrollbar-thumb 滚动条里面的小方块，能向上向下移动（或往左往右移动，取决于是垂直滚动条还是水平滚动条）

::-webkit-scrollbar-track 滚动条的轨道（里面装有Thumb）

::-webkit-scrollbar-button 滚动条的轨道的两端按钮，允许通过点击微调小方块的位置。

::-webkit-scrollbar-track-piece 内层轨道，滚动条中间部分（除去）

::-webkit-scrollbar-corner 边角，即两个滚动条的交汇处

::-webkit-resizer 两个滚动条的交汇处上用于通过拖动调整元素大小的小控件

## JS数组去重封装

         // 数组去重封装

        ArrayDuplication(arr) {

            for (var i = 0; i < arr.length; i++) {

                for (var j = i + 1; j < arr.length; j++) {

                    if (arr[i] == arr[j]) {

                        //如果第一个等于第二个，splice方法删除第二个

                        arr.splice(j, 1);

                        j--;

                    }

                }

            }

            return arr; // 使用return向外提供接口

        }

## CSS三角形绘制

**前言**

**注释：没有设置哪一条边就指向哪一条边**

三角形实现原理：宽度width为0；height为0；（1）有一条横竖边（上下左右）的设置为border-方向：长度 solid red，这个画的就是底部的直线。其他边使用border-方向：长度 solid transparent。（2）有两个横竖边（上下左右）的设置，若斜边是在三角形的右边，这时候设置top或bottom的直线，和右边的斜线。若斜边是在三角形的左边，这时候设置top或bottom的直线，和左边的斜线。

**二、实现**

#### **2.1 Triangle Up**



#triangle-up {

width: 0;

height: 0;

border-left: 50px solid transparent;

border-right: 50px solid transparent;

border-bottom: 100px solid red;

}

#### **2.2 Triangle Down**



#triangle-down {

width: 0;

height: 0;

border-left: 50px solid transparent;

border-right: 50px solid transparent;

border-top: 100px solid red;

}

#### **2.3 Triangle Left**



#triangle-left {

width: 0;

height: 0;

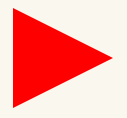
border-top: 50px solid transparent;

border-right: 100px solid red;

border-bottom: 50px solid transparent;

}

#### **2.4 Triangle Right**



#triangle-right {

width: 0;

height: 0;

border-top: 50px solid transparent;

border-left: 100px solid red;

border-bottom: 50px solid transparent;

}

#### **2.5 Triangle Top Left**



#triangle-topleft {

width: 0;

height: 0;

border-top: 100px solid red;

border-right: 100px solid transparent;

}

#### **2.6 Triangle Top Right**



#triangle-topright {

width: 0;

height: 0;

border-top: 100px solid red;

border-left: 100px solid transparent;

}

#### **2.7 Triangle Bottom Left**



#triangle-bottomleft {

width: 0;

height: 0;

border-bottom: 100px solid red;

border-right: 100px solid transparent;

}

#### **2.8 Triangle Bottom Right**



#triangle-bottomright {

width: 0;

height: 0;

border-bottom: 100px solid red;

border-left: 100px solid transparent;

}

#### **空心的三角形呢同理，在当前的三角形后面添加一个一个实心三角形，然后将这个三角形绝对定位到当前三角行的位置切割（记得给父定位）**

.jiao:after{

content: '';

position: absolute;

top: -89px;

left: 2px;

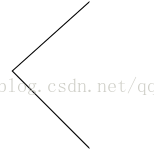
border-top: 89px solid transparent;

border-right: 99px solid #FFFFFF;

border-bottom: 99px solid transparent;

}

效果：



## CSS阴影

基本用法





box-shadow:2px 2px 5px #000;



box-shadow:0px 0px 10px #000;

内阴影



box-shadow:inset 2px 2px 5px #000;

阴影扩展长度值



box-shadow:0px 0px 5px 10px #000;

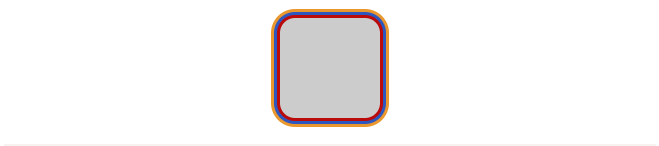


box-shadow:0px 15px 10px -15px #000;



box-shadow:inset 0px 15px 10px -15px #000;

多重阴影



box-shadow:0px 0px 0px 3px #bb0a0a,

0px 0px 0px 6px #2e56bf,

0px 0px 0px 9px #ea982e;

伪元素::before和::after的乐趣

使用伪元素::before和::after，我们能创造出非常逼真的只有图片才能实现的阴影效果。让我来看一个例子：



.box11 {

    width: 300px;

    height: 100px;

    background: #ccc;

    border-radius: 10px;

    margin: 10px;

}

.shadow {

    position: relative;

    max-width: 270px;

    box-shadow: 0px 1px 4px rgba(0,0,0,0.3),

                0px 0px 20px rgba(0,0,0,0.1) inset;

}

.shadow::before,

.shadow::after {

   content:"";

   position:absolute;

   z-index:-1;

}

.shadow::before,

.shadow::after {

   content:"";

   position:absolute;

   z-index:-1;

   bottom:15px;

   left:10px;

   width:50%;

   height:20%;

}

.shadow::before,

.shadow::after {

   content:"";

   position:absolute;

   z-index:-1;

   bottom:15px;

   left:10px;

   width:50%;

   height:20%;

   box-shadow:0 15px 10px rgba(0, 0, 0, 0.7);

   transform:rotate(-3deg);

}

.shadow::after{

   right:10px;

   left:auto;

   transform:rotate(3deg);

 }

## 组件封装

HTML

                <table class="table">

                    <thead>

                        <tr>

                            <th v-for="(item,index) in tableData" :key="index">{{ item.Title }}</th>

                        </tr>

                    </thead>

                    <tbody>

                        <tr v-for="(item,index) in dataSlot" :key="index">

                            <td v-for=" (i ,ikey) in  item" :key="ikey">{{ i }}</td>

                        </tr>

                    </tbody>

                </table>

                <slot></slot>

Data

            tableData: [

                {

                    Title: '用户名',

                    key: 'slot1'

                },

                {

                    Title: '行业',

                    key: 'slot2'

                },

                {

                    Title: '企业名称',

                    key: 'slot3'

                },

                {

                    Title: '手机号码',

                    key: 'slot4'

                },

                {

                    Title: '累积浏览次数',

                    key: 'slot5'

                },

                {

                    Title: '最后浏览时间',

                    key: 'slot6'

                }

            ],

            dataSlot: [

                {

                    slot1: '奥术大师多',

                    slot2: '奥术大师多',

                    slot3: '奥术大师多',

                    slot4: '奥术大师多',

                    slot5: '奥术大师多',

                    slot6: '奥术大师多'

                },

                {

                    slot1: '奥术大师多',

                    slot2: '奥术大师多',

                    slot3: '奥术大师多',

                    slot4: '奥术大师多',

                    slot5: '奥术大师多',

                    slot6: '奥术大师多'

                }

            ]

## Webpack配置域名

在Min.js中配置

// 配置请求域名

import axios from 'axios'; //导入axios

axios.defaults.baseURL = 'https://api.gzwjfh.com/'; //配置根域名

Vue.prototype.$axios = axios; //把axios挂载到Vue的原型上

## Webpack配置跨域