Vue实现选座界面核心功能

在前端界面开发中，有很多界面效果使用原生的JS和html、css实现起来是比较复杂的，但是我们可以使用框架来解决一些比较复杂的功能，在这里我们选择使用Vue来实现选座界面功能的开发，首先我们需要先了解一下需求图1，图1是一个选座界面，当我们看到这个界面之后，会发现涉及到的功能是比较多的，比如拖拽、放大等效果，这样的话我们就可以对选座页面进行功能划分之后再进行开发，首先我们需要从界面效果上先去进行分配界面呈现的效果。

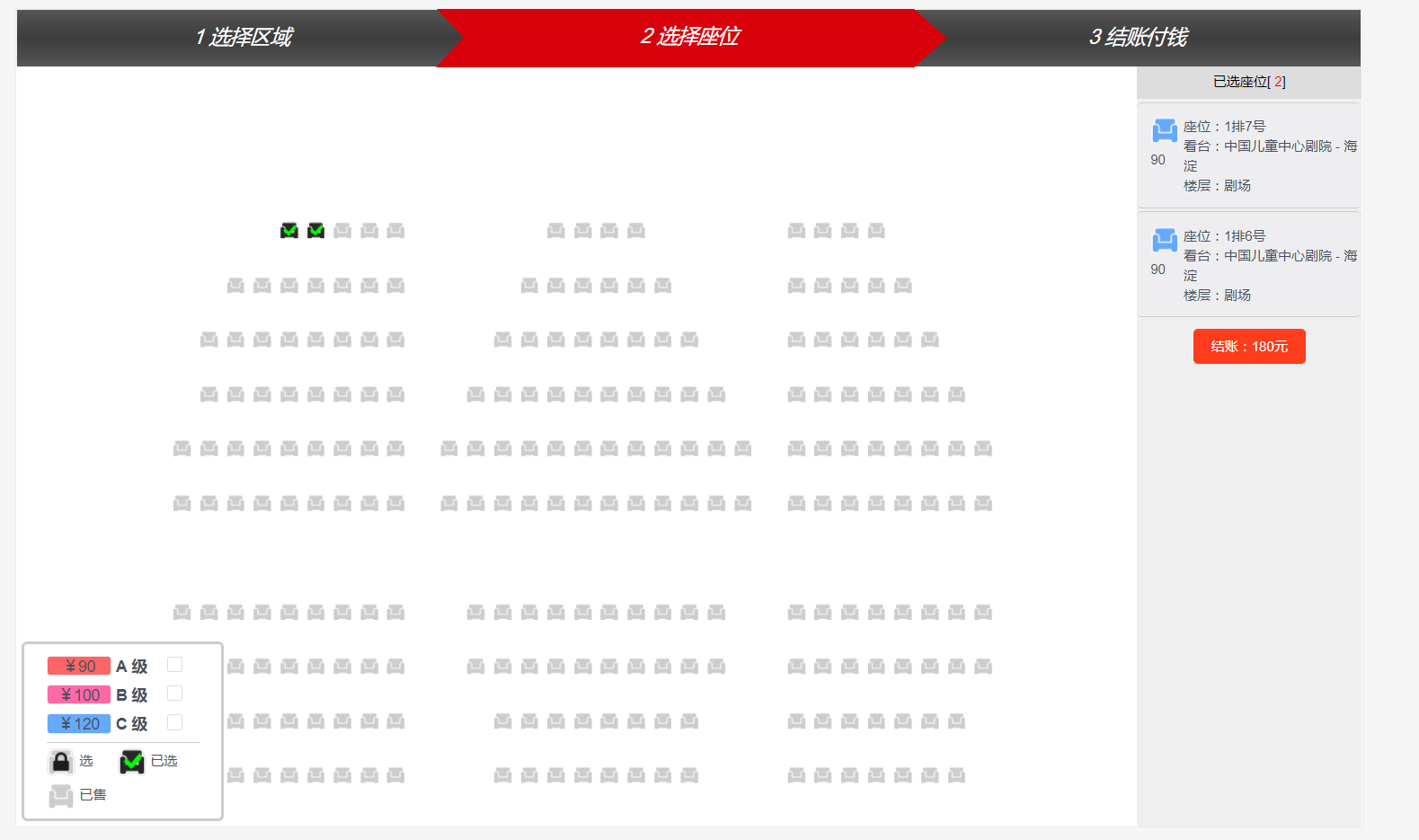


图1

## 选座骨架搭建

界面效果划分为头部，中间选座区，右侧结算区，左下角图例区，这样的话我们就可以在代码中先去划分上这四部分的界面数据了， 具体的结构代码如下：

<template>

<div class="choose-seat-container" >

<div class="choose-seat-head">

</div>

<div id=” seatMap” class="'choose-seat-body ">

</div>

<div class="choose-seat-right " >

</div>

<div class="choose-seat-bottom ">

</div>

</div>

</template>

根据现有的结构我们可以将其分为四个不同的区域实现position进行定位移动对应的位置

CSS（Less语法）：

@top: 50px;

.choose-seat-container {

border: 1px solid #eee;

overflow: hidden;

position: relative;

.choose-seat-head {

height: 50px;

width: 100%;

position: absolute;

border: 1px solid #eeeeee;

top: 0;

}

.choose-seat-right {

width: 200px;

// height: 100%;

position: absolute;

background-color: #efefef;

overflow: auto;

text-align: center;

right: 0;

top: @top;

}

.choose-seat-bottom {

width: 180px;

height: 160px;

position: absolute;

background-color: #ffffff;

border-radius: 5px;

border: 2px solid #ccc;

bottom: 5px;

left: 5px;

padding-top: 5px;

}

.choose-seat-body {

top: @top;

padding-top: 130px;

width: 1000px;

position: absolute;

overflow: hidden;

}

}

这样我们就可以对这四块区域进行逐级添加内容了，首先我们需要添加头部信息，头部信息是非常好添加的不需要依赖外部的数据，我们直接使用div标签添加就可以了具体的代码如下：

<div class="choose-seat-head">

<div class="head-first">1 选择区域</div>

<div class="head-second">2 选择座位</div>

<div class="head-third">3 结账付钱</div>

</div>

这个界面涉及到比较复杂的CSS样式代码具体代码如下：

@red: #d8000b;

.head-second {

background: #d8000b;

position: relative;

height: 52px;

top: -1px;

}

.head-second:before {

content: "";

position: absolute;

display: block;

top: -1px;

width: 60px;

border: 25px solid @red;

border-bottom: 27px solid @red;

border-top: 26px solid @red;

height: 50px;

z-index: 1;

}

.head-second:after {

content: "";

position: absolute;

display: block;

top: -1px;

width: 60px;

border: 27px solid @red;

border-left: 30px solid @red;

height: 54px;

z-index: 1;

}

.head-second:before {

left: -26px;

border-left-color: transparent;

}

.head-second:after {

right: -60px;

border-top-color: transparent;

border-bottom-color: transparent;

border-right-color: transparent;

}

## 选座区缩放拖拽实现

这样我们的第一部分头部信息就添加完成了，接下来我们做第二部分中间选座区的界面显示，首先我们需要先做这样的两步，第一步是选座区增加一个可拖拽和放大的事件，需要在JS脚本里面添加一个方法，具体代码如下：

methods: {

bindScroll() {

// 添加鼠标点下和移动事件，通过这两个事件进行监听，通过修改position的top和left值进行界面的移动。

var oImg = document.getElementById("seatMap");

oImg.onmousedown = function(ev) {

var ev = ev || event;

var disX = ev.clientX - oImg.offsetLeft;

var disY = ev.clientY - oImg.offsetTop;

if (oImg.setCapture) {

oImg.setCapture();

}

document.onmousemove = function(ev) {

var ev = ev || event;

oImg.style.left = ev.clientX - disX + "px";

oImg.style.top = ev.clientY - disY + "px";

};

document.onmouseup = function() {

document.onmousemove = document.onmouseup = null;

//释放全局捕获 releaseCapture();

if (oImg.releaseCapture) {

oImg.releaseCapture();

}

};

return false;

};

// 监听滚轮滚动事件，并使用scale 对页面进行操作，实现界面的放大和缩小。

oImg.onmousewheel = fn;

if (oImg.addEventListener) {

oImg.addEventListener("DOMMouseScroll", fn, false);

}

var scaleSize = 1;

function fn(ev) {

var ev = ev || event;

var b = true;

if (ev.wheelDelta) {

b = ev.wheelDelta > 0 ? true : false;

} else {

b = ev.detail < 0 ? true : false;

}

if (b) {

scaleSize += 0.03;

scaleSize = scaleSize > 2 ? 2 : scaleSize;

} else {

scaleSize -= 0.03;

scaleSize = scaleSize < 0.5 ? 0.5 : scaleSize;

}

this.style.transform = "scale(" + scaleSize + ")";

if (ev.preventDefault) {

ev.preventDefault();

}

return false;

}

}

}

## 选座区域渲染

这样我们对应的界面就可以进行放大和缩小也可对页面进行移动了。第二步是进行选座信息的添加，这部分信息主要是针对给出的数据进行分配，比如我们传入的数据对象seatOrder格式如下： [

"\_\_\_\_\_aaaaa\_\_\_\_\_aaaa\_\_\_\_\_aaaa\_\_\_\_",

"\_\_\_ccccccc\_\_\_\_cccccc\_\_\_\_ccccc\_\_\_",

"\_\_aaaaaaaa\_\_\_aaaaaaaa\_\_\_aaaaaa\_\_",

"\_\_cccccccc\_\_bbbbbbbbbb\_\_ccccccc\_",

"\_aaccccccc\_bbbbbbbbbbbb\_ccccccca",

"\_aaccccccc\_bbbbbbbbbbbb\_ccccccca",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_",

"\_aaaaaaaaa\_\_cccccccccc\_\_aaaaaaaa",

"\_aaaaaaaaa\_\_cccccccccc\_\_aaaaaaaa",

"\_\_aaaaaaaa\_\_\_aaaaaaaa\_\_\_aaaaaaa\_",

"\_\_aaaaaaaa\_\_\_aaaaaaaa\_\_\_aaaaaaa\_"

]

根据这组数据，我们可以对其进行编译和处理，因为我们这里设置的座位信息是分为abc三种的，那我们可以在设置座位信息的时候就可以对这些数据进行一些样式甚至坐标的控制，比如按座位的类型分为：seat-area-\*、seat-gray-\*、seat-charts-\*这类的样式都分为abc三种，如果后期再增加其他样式效果也是比较方便添加的。

得到这样的数组列表数据之后，我们该如何对此数据进行遍历呢？这个是比较关键的一个问题，我们从这个数组中可以看出，其中可以看做是一个二维数组，此数组本身的行数可以代表为总共有多少行座位，每组数据中的\_abc这些信息可以看做是每行数据中的作为其中”\_”代表的是空位置，在界面显示时是没有座位信息的，那我们就可以这样去设计界面的显示了代码如下：

<div id="seatMap" :class="'choose-seat-body '+ chartsSeatArea.join(' ') ">

<div class="seat-charts-row" v-for="(item,x) in seatOrder" >

<div v-for="(subItem,y) in getItems(item)" >

<div v-if="subItem!='\_'" @click="clickCell" :ref="(x+1)+'\_'+(y+1)" :class="'seat-charts-cell seat-charts-seat '+icons[subItem] "></div>

<div v-if="subItem=='\_'" :class="'seat-charts-cell '+'seat-charts-'+icons[subItem] ">

</div>

</div>

</div>

</div>

在此段代码中有一个getItems方法，这个方法是对字符串进行拆分之后返回一个数组对象的，这样我们在遍历列值时就比较方便了，具体的getItems方法代码如下：

getItems(item) {

return item.match(/[a-z\_]{1}(\[[0-9a-z\_]{0,}(,[0-9a-z\_ ]+)?\])?/gi);

},

## 座位事件绑定

到这一步我们对于的选座就渲染完成了，虽然渲染完成了但是我们还需要添加一些事件，例如点击事件，区域选择事件等等，为方便后期操作我们定义了这样几个方法：

hasClass(el, className) {

let classList = el.classList;

let flag = false;

for (let i = 0; i < classList.length; i++) {

if (className == classList[i]) {

flag = true;

}

}

return flag;

},

addClass(el, className) {

el.classList.add(className);

},

removeClass(el, className) {

el.classList.remove(className);

},

toogleClass(el, className) {

let classList = el.classList;

let flag = this.hasClass(el, className);

if (flag) {

/\*

调用删除操作 需要添加数组样式删除代码，如果调用报错，放开下列代码

删除数组中的数据信息，仅限字符串 String 类型的数组

Array.prototype.indexOf = function(val) { for (var i = 0; i < this.length; i++) { if (this[i] == val) return i; } return -1; };

Array.prototype.remove = function(val) { var index = this.indexOf(val); if (index > -1) { this.splice(index, 1); } };

\*/

classList.remove(className);

} else {

/\* 调用添加操作 \*/

classList.add(className);

}

},

hasClass(el, className)判断el元素是否包含className样式，包含返回true不包含返回false；addClass(el, className) 给el元素添加一个className样式，removeClass(el, className) 删除el元素上的className这个样式；toogleClass(el, className)判断el元素上是否有className样式，如果有此样式则删除，如果没有则添加此样式；

当我们鼠标点击此节点的时候我们就可以调用点击事件去触发，这样我们点击的时候就可以给添加上对应的样式信息了，具体的点击事件代码如下：

clickCell(el) {

this.toogleClass(el.currentTarget, "available");

},

.available {// 激活后的样式信息

background-image: url("img/sofa-selected.png") !important;

}

## 座位内容填充

这样我们就绑定上了对应的事件了，但是我们需要绑定的是具体的业务数据，哪些座位是有坐的，哪些是没有坐的，我们通过props将数据传输到当前的组件，其中data是传入的选座的数据，我们需要针对这个data参数进行进一步的数据处理，其中data对象的数据格式为：

{

scheduleId: 12,

cinemaId: 13,

seatPriceList: [

{ areaLevelName: "A", price: 90 },

{ areaLevelName: "B", price: 100 },

{ areaLevelName: "C", price: 120 }

],

seatInfoList: [

{

id: "12\_12",

x: 1,

y: 6,

areaLevel: "A",

cinemaId: "12",

sort: "16",

status: 1

},

{

id: "12\_2",

x: 1,

y: 7,

areaLevel: "A",

cinemaId: "12",

sort: "16",

status: 1

}

]

}

其中seatPriceList为对应区域的价格，seatInfoList为对应的座位信息其中，status代表状态，有数据的就对其座位进行展示，并根据状态控制是否能点击进行选座。

具体的处理逻辑我是直接给$data（Vue原生对象）增加了两个变量，通过自调用函数的方式进行数据处理，具体代码如下：

seatLevelPrice: (param => {

let priceMap = new Map();

for (let i = 0; i < param.length; i++) {

let item = param[i];

priceMap[item.areaLevelName] = item.price;

}

return priceMap;

})(this.data.seatPriceList),

seatData: (param => {

let seatMap = {};

for (let i = 0; i < param.length; i++) {

let item = param[i];

seatMap[item.x + "\_" + item.y] = item;

}

// console.log(seatMap);

return seatMap;

})(this.data.seatInfoList)

得到数据之后我们就可以对界面进行进一步操作了，具体HTML代码如下：

<div id="seatMap" :class="'choose-seat-body '+ chartsSeatArea.join(' ') ">

<div class="seat-charts-row" v-for="(item,x) in seatOrder" :style="'width:'+getItems(item).length\*25+'px;'">

<div v-for="(subItem,y) in getItems(item)" :class="subItem!='\_' && seatData[(x+1)+'\_'+(y+1)] ?('seat-charts-cells seat-area-'+subItem):(subItem!='\_'?('seat-charts-spaces seat-gray-'+subItem):'seat-charts-spaces')">

<div v-if="subItem!='\_' && seatData[(x+1)+'\_'+(y+1)] " :id="(x+1)+'\_'+(y+1)" :ref="(x+1)+'\_'+(y+1)" @click="clickCell" :class="'seat-charts-cell seat-charts-seat '+icons[subItem] ">

<Tooltip placement="top">

&nbsp; &nbsp;

<div slot="content" style="z-index:10000;">

<p>座位：{{(x+1)}}排{{(y+1)}}号</p>

<p>楼层：剧场</p>

<p>看台：{{adress}}</p>

<p style="text-transform: uppercase;">等级：{{seatLevelPrice[subItem.toLocaleUpperCase()]}} {{subItem}}区</p>

<p>票价：{{seatLevelPrice[subItem.toLocaleUpperCase()]}} 元</p>

<p>状态：{{seatData[(x+1)+'\_'+(y+1)].status==1?'可售':'不可售'}}</p>

</div>

</Tooltip>

</div>

<div v-if="subItem!='\_' && !seatData[(x+1)+'\_'+(y+1)]" :id="x+'\_'+y" :ref="x+'\_'+y" :class="'seat-charts-cell seat-charts-seat ' ">

</div>

<div v-if="subItem=='\_'" :class="'seat-charts-cell '+'seat-charts-'+icons[subItem] ">

</div>

</div>

</div>

</div>

其中Tooltip 为iView组件，这里直接调用就可以了，其中的参数是当前座位的信息，是通过props : ["seatOrder", "data", "adress", "seatPojo"]，得到了关于adress、seatPojo 的信息，数据联动我们使用的是x，y坐标进行控制的，数据在存储时也是根据x，y坐标进行存储的。

## 6.选座业务数据处理

这样界面显示的信息就算完成了， 主体选座信息就可以正常显示了，但是当我们点击之后，我们右侧需要显示对应的选座信息，其实，这块的显示还是比较简单的，比如我们可以定义一个变量叫selectedMap，这个selectedMap 我们可以在点击的时候给其添加上对应的数据，等再点击的时候删除对应的数据，具体的selectedMap添加与删除的方法如下：

clickCell(el) {

this.toogleClass(el.currentTarget, "available");

let id = el.currentTarget.id;

let item = this.seatData[id];

item.price = this.seatLevelPrice[item.areaLevel];

if (this.selectedMap.get(id)) {

this.selectedMap.delete(id);

} else {

this.selectedMap.set(id, item);

}

this.size = this.selectedMap.size;

this.price = 0;

this.selectedMap.forEach(item => {

this.price += item.price;

});

},

我们在渲染座位信息时就讲id添加到座位上了， :id="(x+1)+'\_'+(y+1)"这里使用的是x，y坐标点进行控制的。selectedMap的对象类型是一个Map是ES6中新的语法，需要大家掌握Map的具体使用。

当我们得到这个对象之后我们就能非常简单的渲染右侧的列表了，具体页面代码如下：

**HTML代码:**

<div id="seatMap" v-if="size>0" class="choose-seat-right " :style="'height:' + (containerHeight -50 ) + 'px;'">

<div class="selected-seat-count">

已选座位[

<span>{{size}}</span>]

</div>

<div class="selected-seat-desc" v-for="item in this.selectedMap.toJSON()">

<div class="select-seatDesc-left">

<i class="sofa-blue"></i>

<br> {{item[1]["price"]}}

</div>

<div class="selected-seatDesc-right">

<p>

座位：{{item[1].x}}排{{item[1].y}}号

</p>

<p> 看台：{{adress}}</p>

<p>

楼层：剧场

</p>

</div>

</div>

<Button type="primary" @click="submit" style="margin-top:10px;">结账：{{price}}元</Button>

</div>

**CSS样式：**

.choose-seat-right {

width: 200px;

// height: 100%;

position: absolute;

background-color: #efefef;

overflow: auto;

text-align: center;

right: 0;

top: @top;

.selected-seat-count {

background: #dddddd;

color: #000;

width: 100%;

height: 30px;

line-height: 30px;

text-align: center;

font-weight: 600px;

span {

color: red;

}

}

.selected-seat-desc {

background: #eeeef1;

padding: 12px 3px 10px 12px;

border-top: 1px solid #ccc;

border-bottom: 1px solid #ccc;

border-radius: 5px;

margin-top: 3px;

overflow: hidden;

text-align: left;

.select-seatDesc-left,

.select-seatDesc-right {

display: inline-block;

float: left;

height: 70px;

}

.select-seatDesc-right {

margin-left: 8px;

line-height: 25px;

}

}

}

显示控制我们采用变量值的方式显示，就是获取selectedMap的大小，当其值个数大于0才会显示出右侧的界面，这样在显示的时候就非常灵活的控制了。

到这里我们的选座页面基本接近尾声了，但是还有一个功能需要增加上就是图例界面效果，图例代码比较简单我们来添加一下，图例代码如下：

**HTML代码：**

<div id="seatMap" class="choose-seat-bottom ">

      <div class="seat-area-legend">

        <CheckboxGroup v-model="chartsSeatArea">

          <div class="seat-area-row" v-for="item in data.seatPriceList">

            <h2 :class="'cell-'+item.areaLevelName">￥{{item.price}}</h2>

            <strong>{{item.areaLevelName}} 级</strong>

            <Checkbox style="float:right;" data-id="item.areaLevelName" :label="'chartsSeat-area-'+item.areaLevelName">&nbsp;</Checkbox>

          </div>

        </CheckboxGroup>

        <div class="sofa-legend">

          <div>

            <i class="sofa-locked"></i> 选

          </div>

          <div>

            <i class="sofa-selected"></i> 已选

          </div>

          <div>

            <i class="sofa-gray"></i> 已售

          </div>

        </div>

      </div>

    </div>

**CSS代码：**

.choose-seat-bottom {

    width: 180px;

    height: 160px;

    position: absolute;

    background-color: #ffffff;

    border-radius: 5px;

    border: 2px solid #ccc;

    bottom: 5px;

    left: 5px;

    padding-top: 5px;

    .seat-area-row {

      margin: 5px 20px;

      h2 {

        width: 58px;

        height: 18px;

        padding: 1px 5px;

        display: inline-block;

        text-align: center;

        border-radius: 3px;

        line-height: 15px;

        border: 1px solid #eeeeee;

      }

      .cell-A {

        background: #fd6566;

      }

      .cell-B {

        background: #fd68a6;

      }

      .cell-C {

        background: #65a9fd;

      }

    }

    .seat-area-row:last-child {

      padding-bottom: 5px;

      border-bottom: 1px solid #ccc;

    }

    .sofa-legend {

      margin: 5px 20px;

      div {

        display: inline-block;

        width: 60px;

        height: 25px;

        margin-bottom: 5px;

        i,

        span {

          vertical-align: middle;

        }

      }

    }

  }

从代码上看还是比较简单的，但是当我们点击一个组的时候，整体组成员就会显示出不同的样式，这样的话我们如何操作呢？ 其实比较简单，我们只需要增加这样的一组CheckBox，这里我们使用iview的CheckboxGroup 组件，使用v-model="chartsSeatArea"绑定这个数据，我们渲染时将样式给到这个CheckBox的值上，然后通过点击选中返回对应的数据对象，然后在<div id="seatMap" :class="'choose-seat-body '+ chartsSeatArea.join(' ') ">增加上这段样式，使用数组的join方式将其拼接起来。

到这里基本上这个选座界面就完成了，后期需要将数据传输到下一个界面显示了。